ffice of cademic omputing

CONTROLLING NSW TOOLS AND CONFIGURATIONS
UNDER OS/MVT

NSW Semi-Annual Technical Reports

January 1, 1979 - June 30, 1979
and
July 1, 1979 - December 31, 1979

UCLA TR-24

Neil Ludlam
Denis De La Roca

niversity of alifornia, os ngeles

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: OAC/TR24	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: Controlling NSW Tools and Configurations Under OS/MVT National Software Works	5. TYPE OF REPORT & PERIOD COVERED: Semi-Annual 1/1/79 - 12/31/79
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): D. DeLa Roca; N. Ludlam	8. CONTRACT OR GRANT NUMBER(s): MDA 903-74-C-0083
9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of California, Office of Academic Computing, Los Angeles, CA 90024	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Prog. Elemt: 62708E; Program Code: 0T10; ARPA Order No. 2543
11. CONTROLLING OFFICE NAME AND ADDRESS: Defense Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 2209, Attn: Program Mgt. OFfice	12. REPORT DATE: 12/1/80
	13. NUMBER OF PAGES: 147
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*: Defense Supply Service - Washington, Room 1D-245, The Pentagon, Washington, D.C. 20310, R. Mueller	15. SECURITY CLASS. *(of this report)*
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

~~Distribution Unlimited~~

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

National Software Works, NSW, ARPANET, NSW tools, configuration management, FOREMAN, encapsulator, command interpreter, batch job package.

UNIVERSITY of CALIFORNIA (UCLA)

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report covers technical development at UCLA relating to the National Software Works (NSW) during 1979. It is primarily concerned with the design implementation and configuration of mechanisms required to execute "tool" programs at UCLA under NSW, i.e., a FOREMAN for interactive tools, and a BATCH JOB PACKAGE for batch tools.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

CONTROLLING NSW TOOLS AND CONFIGURATIONS UNDER OS/MVT

by
Neil Ludlam
Denis De La Roca

December 1, 1980

Document TR-24

UCLA Office of Academic Computing
5628 Math Sciences Addition
University of California C0012
Los Angeles, California 90024

This work was sponsored by
the Advanced Research Projects Agency
of the Department of Defense,
under ARPA Order no. 2543,
Contract No. MDA 903-74-C-0083:

ARPANET COMPUTER SERVICES IN SUPPORT OF
THE NATIONAL SOFTWARE WORKS

June 1, 1975 - February 29, 1980

William B. Kehl, Principal Investigator
(213) 825-7511

SEMI-ANNUAL TECHNICAL REPORTS
for period of
January 1, 1979 - December 31, 1979

REPORT SUMMARY

This report covers technical development at UCLA relating to the National Software Works (NSW) during 1979. It is a combination of the two Semi-annual Technical Reports covering the periods of January 1 through June 30 and July 1 through December 31 of 1979.

The primary goal of the NSW project at UCLA is to make the IBM Operating System OS/MVT, and specifically its implementation on the UCLA IBM 3033, a "tool-bearing host" within the NSW.

This report is primarily concerned with the design and implementation of mechanisms required to execute "tool" programs at UCLA under NSW. These mechanisms are: a "Foreman" to supervise execution of interactive tools; a "Batch Job Package" to supervise execution of batch tools; and tools themselves.

A secondary concern in this report is the problem of configuration management in the NSW system.

The subsequent sections of this report correspond to documents stored in the NSW documentation repository maintained by the NSW Operations Contractor, so each section has been made self-contained. For example, each section has its own table of contents and reference summary, and each section is independently paginated.

Part II: FM/360 -- The NSW MVT Foreman

> This section describes FM/360, the Foreman implementation for OS/MVT, from the aspect of its use as an NSW core-system component.

Part III: The UCLA Encapsulator Command Interpretor

> This section describes that subcomponent of FM/360 called the "Encapsulator Command Interpretor", or ECI. The ECI can be viewed as a separable piece of software that interprets a simple program of statements that constitute a local interactive tool descriptor. Interpretation of this program accomplishes the gathering of information from the user, the setting up of input files, the execution of the tool programs themselves, and the disposition of output files.
>
> The separation of function between the Foreman and the ECI is not complete, however, it serves the useful purpose of breaking the rather massive Foreman down into two more easily described parts.

Part IV: BJP/360 -- The MVT Batch Job Package

This section describes BJP/360, the NSW Batch Job Package as implemented for OS/MVT. The BJP performs operations that are not supported by OS/MVT except through extensive local system modifications. Such modifications must be made independently by each OS/MVT installation, so that the BJP is considerably less readily exportable than other OS/MVT NSW components.

Part V: Configuration Management

This section examines the problem of configuration management under OS/MVT. It explores the options available using existing mechanisms, and proposes some simple additional mechanisms that might aid in configuration management.

PART II

FM/360 -- THE NSW MVT Foreman

This section is separately available
as UCLA document UCNSW-205

2. PART II: FM/360

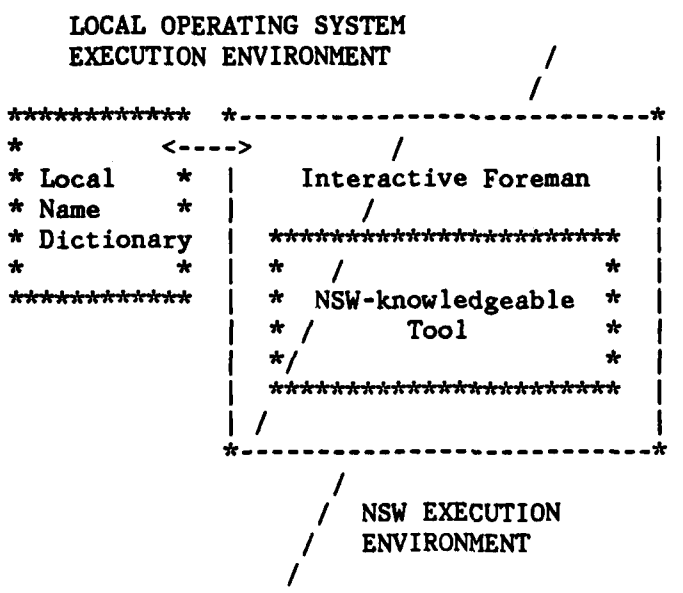
2.1. THE NSW FOREMAN

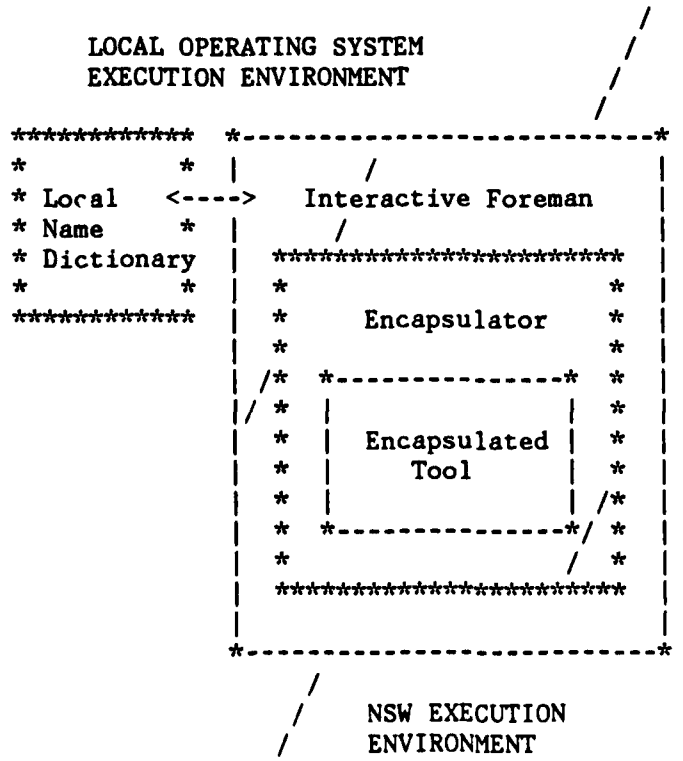
Within the National Software Works (NSW), each Tool-bearing host (TBH) that is to support interactive tools is required to have a software component called a "Foreman" (reference 1). The Foreman is that part of the NSW core system which provides the NSW execution environment, and the NSW monitor services, to the executing tool. It can be viewed as a kind of emulator of the NSW execution environment, running in the execution environment of the local system.

In the case of encapsulated tools (those which are unaware of the NSW environment or services), the Foreman includes an Encapsulator interface, which may be viewed as an emulator for the execution environment in which the tool believes itself to be running, and as an interface converter between that environment and the NSW tool execution environment. These ideal relationships are illustrated in figure 1.

The reader is assumed to be familiar with reference 1, which prescribes the operation and protocols of an NSW Foreman, and with the software environment provided by the NSW.

Figure 1: Foreman Relationship to Tool and NSW





2.2. FM/360 AS A FOREMAN SUBSET

This document describes FM/360, a Foreman implementation for IBM real-memory systems. Specifically, FM/360 was developed to operate on the UCLA IBM System/360 Model 91KK under the MVT Operating System with the Time-Sharing Option, TSO (we commonly refer to this combination as OS/MVT). However, with the modificaton of certain installation - dependent modules, it will operate on any upward - compatible system.

The constraints of IBM Operating Systems, along with the severe limitiation of main storage available in the real-memory system OS/MVT, have combined to make FM/360 a much less comprehensive component than an ideal Foreman should be. Major areas of functionality have had to be omitted, and severe limitiations have been placed on those areas of functionality that survive. The resulting implementation is illustrated in Figure 2.

Specific differences include:

* FM/360 does not maintain a Local Name Dictionary (LND), which is the Foreman's crash-recovery cache. It is thus not possible to recover a tool session if it is terminated by a crash of OS/MVT, of the NSW, or of FM/360 itself.

* Accordingly, those Foreman scenarios dealing with crash recovery are not supported by FM/360, although some of the corresponding transactions are accepted. This includes the "save-LND" and "Rebegintool" scenarios.

* Only encapsulated tools are supported. There is no Tool-to-Foreman monitor-service interface that could be called by a tool that was knowledgeable about NSW services.

* Encapsulation, which should be done through sub-monitoring of the tool process, is simulated by pre- and post-tool-execution use of the "Encapsulator Command Interpretor" (ECI -- see reference 7). The ECI attempts to set up the tool's operating environment in such a way that, as it operates in its native mode, the overall effects are the same as though it were being dynamically monitored.

* FM/360 operates asynchronously to its tool task; however, it shares its protection domain. Therefore, buggy tools can damage the executing FM/360 process. For this reason an instance of FM/360 is never permitted to recycle and monitor a second tool instance. Every instance is required to use an entirely new copy of the FM/360 code.

* Both the tool and FM/360 operate as one TSO swapped job. Since the FM/360 code is large compared to many interactive tools, and since TSO regions tend to be sized to accomodate those tools in native mode, it may sometimes happen that large tools will be unable to get the main storage that they need for execution.

Figure 2: FM/360 Relationship to Tool and NSW

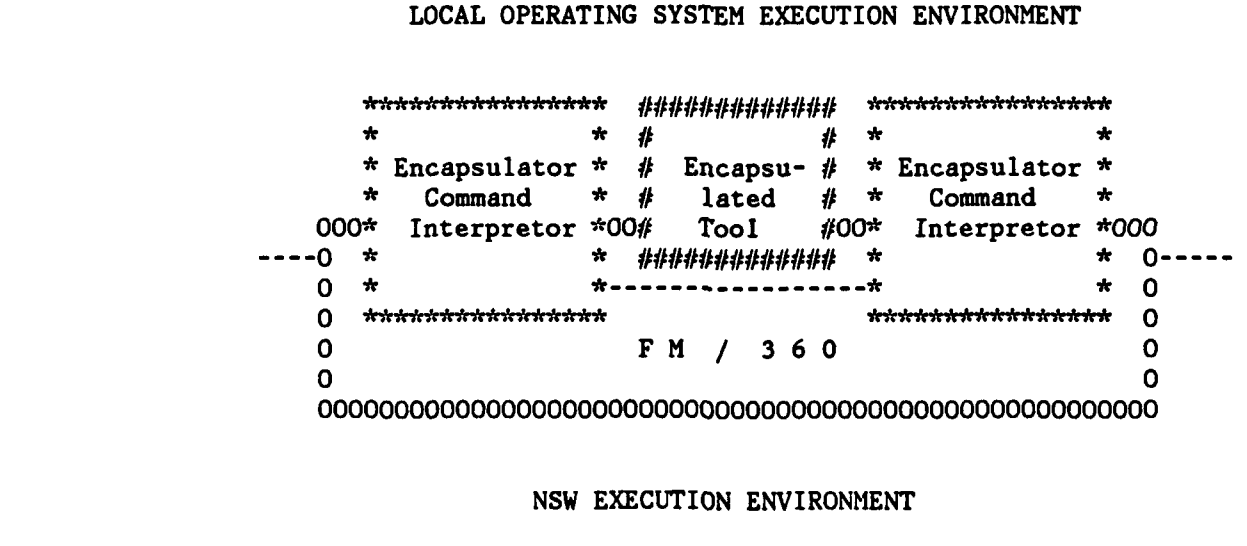
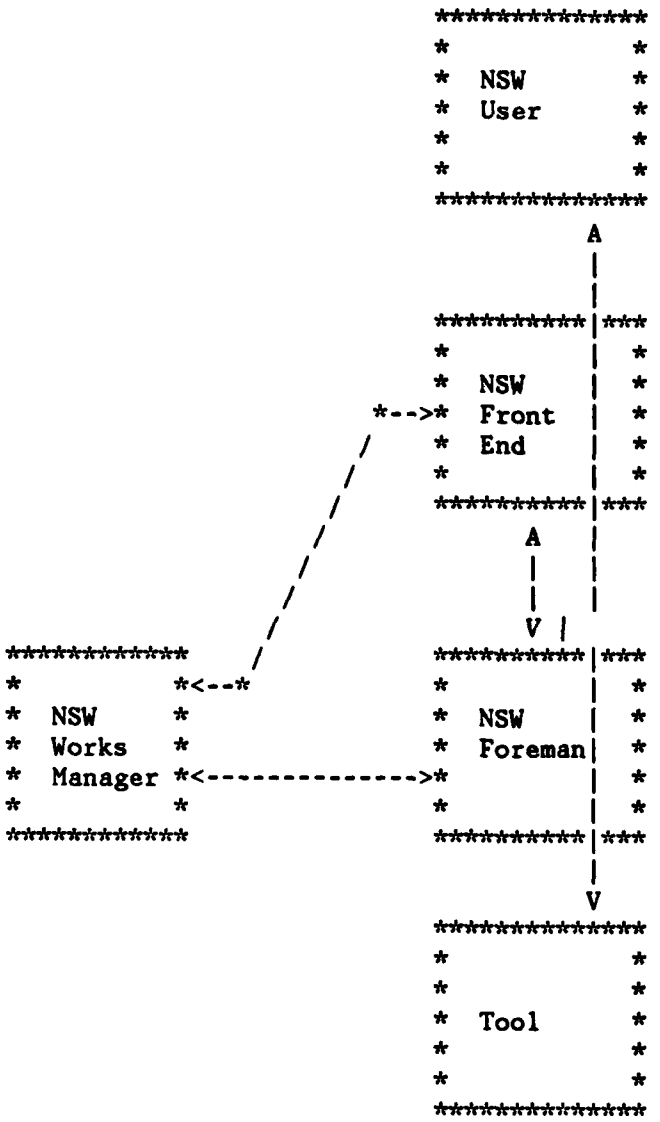


Figure 3: Foreman Communication Paths



2.3. FM/360 COMMUNICATIONS SUMMARY

FM/360 functions as an NSW core-system process with generic name "FOREMAN". It communicates with other NSW processes in a pattern illustrated in Figure 3. One channel of communication, the direct connection between the tool and the user, which is supervised by FM/360, is a standard TELNET (reference 2) network connection. The other channels use the NSW Network Transaction Protocol, or NTP (reference 1, appendix 3), On an IBM system, NTP is implemented on three levels:

* The procedure-call level is implemented by the PL/PCP subroutine package (reference 3).

* The MSG message and direct-connection level is implemented by the PL/MSG subroutine package (reference 4), which also uses the PLOXI package (reference 5).

* The NSWB8 data encodement level is handled by the PL/B8 subroutine package (reference 6).

FM/360 interacts with other NSW processes using well-defined procedure calls, each of which consists of a single NTP transaction of the form "Request/Reply", where the request may be either generically or specifically addressed. Usually, we express such a transaction in the form:

G/process-procedure (arguments) -> (results)

or

S/process-procedure (arguments) -> (results)

or

A/process (alarmnumber) -> (results)

where the three forms correspond to generic addressing, specific addressing, and alarms, respectively. "Process" is a shorthand name for the process that receives and executes the transaction, and "procedure" is the name of the specific procedure that that process is to perform. The "result" may be empty if only a positive or negative acknowledgement is defined. In cases where no response of any kind is expected, the arrow and the "result" are omitted.

2.3.1. TRANSACTIONS GENERICALLY ADDRESSED TO A FOREMAN

An NSW Foreman process should be prepared to process a well-defined set of transactions. FM/360 can receive any of these, but it does not necessarily respond in the way that the ideal Foreman would.

G/FM-BEGINTOOL (description of desired tool)
-> (description of started tool)

The "begintool" transaction establishes a tool instance.

G/FM-REBEGINTOOL (description of interrupted tool)
-> (description of reestablished tool)

The "rebegintool" transaction is intended to resume execution of a tool interrupted by a system crash. FM/360 does not support such restart. Should it ever receive this transaction, it will treat it as though it were the "begintool" transaction.

G/FM-WMOK (to be defined)
-> (to be defined)

The "wmok" transaction is intended to let a Foreman know that the NSW Works Manager has crashed and restarted. It is rejected by FM/360.

2.3.2. TRANSACTIONS SPECIFICALLY ADDRESSED TO A FOREMAN

S/FM-ENDTOOL (type and reason of termination)

The "endtool" transaction is intended to terminate execution of a tool, whether or not the tool itself has terminated.

S/FM-SAVE-LND (reason for termination)
-> (accounting information)

The "save-lnd" transaction is intended to suspend execution of a tool, so that it can be resumed later through "rebegintool".

S/FM-STOPTOOL (optional immediate restart point) ->

The "stoptool" transaction is intended to block the tool task temporarily. It is not implemented by FM/360.

S/FM-STARTTOOL (restart point) ->

The "starttool" transaction is intended to unblock a tool blocked by "stoptool". It is not implemented by FM/360.

S/FM-GUARANTEE (tool identifier)

The "guarantee" transaction is intended to let a Foreman know that all the information it has entered into the NSW core-system data bases is checkpointed, and can safely be removed from the LND.

PA/FM (1)

Alarm code 1 is intended to signal a Foreman that a tool-abort scenario is about to be begun.

A/FM (10) -> (status information)

Alarm code 10 is a request for NSW and Foreman status.

A/FM (11) -> (edited status information)

Alarm code 11 is a request for NSW and Foreman status edited into a user-oriented form. It is rejected by FM/360.

A/FM (12) -> (accounting information)

Alarm code 12 is a request for current accounting information.

A/FM (13) -> (edited accounting information)

Alarm code 13 is a request for current accounting information edited into a user-oriented form. It is rejected by FM/360.

2.3.3. OTHER INCOMING TRANSACTIONS

FM/360 will reject an incoming transaction in any of these cases:

* The calling process is not of generic class "WM" (the NSW Works Manager).
* The requested procedure name or alarm code is unknown.
* The transaction uses the wrong addressing mode (generic vs. specific).
* The transaction parameters violate the expected syntax.
* The alarm code is unknown.

2.3.4. TRANSACTIONS ADDRESSED TO THE FRONT END

Once FM/360 is set onto a task by the Works Manager, much of the rest of its communication is via transactions received from or addressed to the NSW Front End which is controlling the tool-user's console. Since a specific Front End process is assigned to the tool instance, the outgoing Front End transactions are all specifically addressed.

S/FE-OPENCONN (connection description)

The "openconn" transaction negotiates a direct connection between a Foreman and a Front End.

S/FE-TOOLHALTED (accounting information)

The "toolhalted" transaction informs the Front End that tool execution is complete.

2.3.5. TRANSACTIONS ADDRESSED TO A WORKS MANAGER

A Foreman uses transactions addressed to the Works Manager to obtain file-system service for its tool and/or user, and also as a backup resource in case the Front End is not available to gracefully terminate a session. These services are not performed by the Works Manager instance that created the Foreman instance, but by any Works Manager. Therefore, these transactions are generically addressed.

G/WM-TOOLHALTED (accounting information) ->

The "toolhalted" transaction can also be sent to the Works Manager, when contact with the Front End is lost.

G/WM-GET (NSW file description)
-> (local copy description)

A Foreman issues the "get" transaction to obtain access to an NSW file for its tool.

G/WM-DELIVER (local copy description)
-> (NSW file description)

A Foreman issues the "deliver" transaction to move a tool output file back into the NSW file space.

G/WM-LND-SAVED (tool lists)
-> (exceptions)

The "lnd-saved" transaction informs the Works Manager that one or more tool instances have been suspended abnormally.

2.4. PARAMETRIC DATA ELEMENTS

Every Foreman transaction includes a set of parameters encoded in an NSWB8 LIST (reference 6). While the parameter structure of the transaction is peculiar to its procedure, the elements of that structure are commonly defined. This section gives FM/360's interpretation of these elements.

2.4.1. PROGRAM NAME

NTP Transactions that initiate or re-initiate execution of a tool include a "program name" datum represented as a character string. FM/360 interprets this as the name of an ECI program to be extracted from a local ECI program data base identified by the system control language that set up the FM/360 instance. Actual names of executable object programs are contained within ECI programs, and are not known to the WM-resident tool descriptor.

Viewed another way, the "program name" is a pointer to the locally-resident portion of the Works Manager's Tool Descriptor.

2.4.2. TOOL TYPE

The NSW tool type is an "index" datum that is intended to tell a Foreman whether the tool uses NSW transactions, MSG calls, and/or direct connections. Since FM/360 only supports encapsulated tools that use a single TELNET connection, it does not actually examine this datum.

2.4.3. TOOL INSTANCE ID

The "tool instance id" is a 32-bit quantity which serves as a handle onto the Works Manager's internal representation of a tool instance. FM/360 gets it as a part of a tool-initiating transaction, remembers it, and subsequently uses it in outgoing transactions that need to identify the tool. It is not considered to have structure or to be interpretable by the local system.

2.4.4. ENTRY VECTOR INDEX

The "entry vector index" datum supports the notion of a Foreman process as a sub-supervisor for a tool process. Under this notion, the Foreman could conceivably start and stop execution of the tool process, altering its location counter in meaningful ways. The "entry vector" consists of a virtual vector of well known entry point types, for which, corresponding to each tool instance, there exists a real vector of specific entry points. Each slot in the vector corresponds to a type of entry, such as warm-start, cold-start, termination, continue, etc. The datum passed to a Foreman under the name "entry vector index" is actually an index into the entry vector, and thus selects an entry point at which tool

exection is to be started or restarted.

FM/360 has little use for the entry-vector notion. Under OS/MVT, true sub-supervision does not exist. The location counters of executing programs cannot, in general, be altered in ways that were not explicated at the time of compilation of the program. Therefore, FM/360 generally ignores the "entry vector index" datum in a transaction. One exception exists, though it is not a useful one: if the "begintool" transaction includes an "entry vector index" of zero ("do not start tool at all"), then the tool will be set up in every way, but its process will remain blocked. Theoretically, it could then be unblocked through the "starttool" transaction; however, since that transaction is unsupported, the tool will in fact be permanently blocked.

To use FM/360 in the way intended by its designers, it is advisable to set "entry vector index" to 1 ("cold-start entry") for tool-creating transactions and to 4 ("continue") in other transactions. Such settings will continue to operate as expected if and when FM/360 begins to interpret this datum.

2.4.5. PROCESS ADDRESSES

The immediate environment of an instance of FM/360 includes five specifically addressable NSW processes: the Front End, the originating Works Manager (other instances of the Works Manager, such as those addressed in "get" transactions, are generically addressed), the Foreman process itself, the governed tool process, and the "creating process". This last is invariably the same as the Front End process in the current implementation, but it might not be in some future NSW. The specific addresses of these processes are propagated throughout the group in the NTP transactions that create the associations. When one process creates another, as the Works Manager does to the Foreman, the exchange of process names occurs as a natural consequence of the NTP transaction; however, the addresses of other process in the group are carried as data elements within the bodies of the transactions themselves.

FM/360 is specifically aware of its own process, the creating Works-Manager process, and the Front End process. Of these, only the Front End process address is extracted from the body of a transaction. FM/360 always returns a null process name (a bitstring of length zero) as the address of its tool. This is defined to mean that the tool and Foreman share an NSW process. In the case of FM/360, which is not a sub-supervisor, this is appropriate.

2.4.6. TOOL-DEPENDENT PARAMETER LIST

The "tool-dependent Parameter List" (TDPL) consists of a list of character strings, kept in the Works Manager's tool descriptor, and passed to the Foreman when a tool instance is to be created. It is not interpreted by the Works Manager in any way. It should be

considered to be a piece of Foreman data tabulated, for convenience, by the Works Manager.

Early versions of FM/360 used the TDPL to hold the information that is now stored locally in the ECI program. The current implementation does not examine it at all.

2.4.7. START STATE

The "start state" datum is a boolean state flag set by a Foreman to indicate whether its tool is currently running or stopped. It is set in the reply to "begintool" according to whether the "entry vector index" datum was zero or non-zero. However, since non-zero values of "entry vector index" have already been shown to be not useful in FM/360, it seems appropriate to say that a "false" value of "start state" is possible but not useful in FM/360.

2.4.8. WORKSPACE DESCRIPTOR

When FM/360 creates a tool instance, it returns a "workspace descriptor" for the purpose of defining the subspace of the local host file space in which the Works Manager is to build tool copies of NSW files for the tool. This descriptor consists of two character strings, the "workspace name" and the "access information". These two data map directly into the "directory" and "password" data passed to File Package FP/360 (reference 8) when it is operating on behalf of FM/360. The workspace name is always filled with the TSO logon directory under which FM/360 is operating. In the UCLA implementation, this consists of a character string of the form "cccccc.uuu", where "cccccc" is a UCLA "charge number", and "uuu" is a TSO "Userid". This form is not uniform across all MVT/TSO implementations, but the governing rule is this: the workspace is that character string, which, when concatenated ahead of a simple file name, with a period between, and with the entire construct then enclosed in single quotes, results in the fully-qualified file name that is equivalent, under the installation's data-set naming rules, to the simple name. This definition depends on the syntax of MVT/TSO file names as given in Reference 9.

2.4.9. TERMINATION REASON

When FM/360 is reporting that a tool has terminated, or has been terminated, it sends a "termination reason" code. Likewise, when FM/360 receives a transaction instructing it to terminate a tool, it receives a "termination reason" code. If the tool terminates under its own control, FM/360 will send a termination reason code of 4 (direct tool termination). In other cases, it will simply echo the termination reason code sent it in the transaction that caused it to terminate the tool.

2.4.10. TERMINATION SCENARIO

When FM/360 is reporting the demise of a tool, it sends a "termination scenario" datum which clarifies the mechanism by which the demise came about. It has the values:

1 -> The tool halted on its own.

2 -> The tool was ended because of an "endtoool" transaction requesting normal termination.

3 -> The tool was ended because of an "endtoool" transaction requesting abort termination.

2.4.11. TERMINATION TYPE

When FM/360 is asked to terminate a tool, it is given a "termination type" code, instructing it how, or whether, to effect delivery of tool output files. This datum has the values:

1 -> Abort the tool and deliver nothing.

2 -> Perform a delivery dialogue with the user. For each potentially deliverable file, ask the user for a disposition decision.

3 -> Perform automatic delivery, without user aid.

FM/360 uses this dataum to distinguish an "abort" type tool termination scenario from an "end" type scenario, for the purpose of filling in the "termination scenario" datum mentioned above. Otherwise, it ignores it. The form of delivery is coded into the ECI program (the local extension of the Tool Descriptor), and cannot be changed (see reference 7).

2.4.12. FAULT DESCRIPTOR

There are two types of "fault descriptors" in FM/360 transactions: the first kind are appended to all transaction replies by the PL/PCP package (reference 3), which performs transaction management; the second kind is included explicitly in the "endtool" transaction.

The use of PL/PCP fault descriptors by FM/360 is limited to the rejection of unimplemented or undecipherable transactions.

The fault descriptor in "endtool" is always a list of count zero.

2.4.13. ACCOUNTING LIST

Whenever a tool is terminated, a Foreman is expected to get information to the Works Manager which will enable the tool execution to be billed to the proper NSW user. This is usually done in the "endtool" request, whether it be addressed to the Front End (usually) or directly to the Works Manager (when Front End contact is lost), but it can also be done in the "save lnd" transaction. The FM/360 version of the "accounting list" is a NSWB8 (reference 6) structure of the form:

```
LIST(cost=integer,
     components
     =LIST(cpu-seconds
            =LIST(type=index=1,
                  amount=integer),
           connect-minutes
            =LIST(type=index=2,
                  amount=integer),
           i/o-operations
            =LIST(type=index=3,
                  amount=integer)))
```

FM/360 fills in these fields accurately; however, the "cost", a value in cents, may not be exactly what will be billed NSW through the normal cyclic billing system at UCLA. It is a best estimate.

2.4.14. STATUS LIST

In response to a "status alarm", FM/360 generates a "status list" consisting of four state variables:

1) External tool state (index), with values:

   0 -> running.
   1 -> never started.
   4 -> terminated.

2) Internal tool state: always zero.

3) Operating System state: always zero.

4) Program counter: always zero.

2.4.15. USER ID

When FM/360 sends "toolhalted" to the works manager, it uses a syntax defined for use by the Front End. One of the elements of the transaction is the "user id", a datum kept only by the Front End. When FM/360 sends this datum, it always has the value zero.

2.4.16. CONNECTION TYPE

The "connection type" code is used by FM/360 in the "openconn" transaction. It is an index variable which always has the value 1, meaning that a TELNET connection is being requested.

2.4.17. CONNECTION ID

The "connection id" code is used by FM/360 in the "openconn" transaction. It is an index variable which usually has the value 1; however, when FM/360 loses touch with the Front End and attempts to establish a new connection, it will increment the value used for the original connection by 1. This scheme eliminates the possibility that the Front End will not yet have noticed the broken connection, and will thus believe that the request is a duplicate.

2.4.18. ATTRIBUTE CODE

The "attribute code" is used in the "get" and "deliver" transactions, which are always issued as a result of interpretation of an ECI statment. Most of the arguments to those transactions are calculated in or coded into the ECI program. It is useful to think of the ECI program as an extension of the Tool Descriptor kept by the Works Manager. In this context, the Tool Descriptor contains a vector of Global File Type (GFT) lists, (as described in reference 8), and the ECI program contains indices into that vector, each of which is a shorthand name for the list of GFT's itself. The "attribute code" is such an index. It is expanded into the actual list of GFT's by the Works Manager when it is making up the corresponding transaction to the File Package.

2.4.19. FILESPEC

Like "attribute code", the "filespec" argument to the "get" transaction comes from the ECI program; however, it is usually gotten through user query, rather than being coded in. It is a character string that identifies the NSW file being addressed. The "get" also returns a "new filespec", which is possibly altered through disambiguation, in the same form.

2.4.20. SEMAPHORE SET

The "semaphore set" argument is a boolean that requests the Works Manager to set the NSW "semaphore" on a file. Like the "attribute code", it is coded into the ECI program. At this writing, NSW semaphores are not functional, so it is not advisable to set this flag.

2.4.21. HELP DIRECTOR

The "help director" argument is an index that tells the Works Manager how to direct "help" calls if more information is needed, such as in the standard file disambiguation procedure. FM/360 always sets it to zero, meaning to ask the user directly for help.

2.4.22. NSW FILENAME

"Get" and "deliver" return full NSW file names as results. These are character strings which may be considered to be just like "filespecs" except that they are fully disambiguated and have all levels expressed.

2.4.23. LOCAL FILENAME

"Get" returns, and "deliver" expects, a "local file name", which is a character string meaningful to the local file system. In the case of FM/360, it is in the form of an OS/MVT DSNAME (reference 9). It can be a quoted, fully qualified name, or an unquoted name relative to the directory (workspace) in which the FM/360 instance is operating.

2.4.24. ENTRY NAME

For the purposes of FM/360, an "entry name" can be considered to be just another "filespec". NSW distinguishes between the two because of different rules for disambiguation.

2.4.25. REPLACE ENABLE

The "replace enable" argument to "deliver" tells the Works Manager whether an existing file with the same name is to be replaced. Like "semaphore set", this always comes from the ECI program.

2.4.26. HERALD

Whenever NSW processes wish to perform a formal handshake, they exchange messages containing "heralds". A herald is just a character string identifying the sender. In this implementation, FM/360 always sends a herald of:

FM/360 HERALD

and it never examines any incoming herald.

2.5. TRANSACTIONS SUPPORTED

An incoming generic transaction is always delivered to a virgin instance of FM/360. Usually, this will mean that MSG central will create an FM/360 instance, wait for it to materialize its NSW process, and then deliver the transaction-creating message. There are some mechanisms in MSG central as implemented at UCLA wich might cause a virgin FM/360 to be materialized and waiting at the time this transaction is received. When this is the case, the transaction is simply assigned to the waiting process.

An incoming specific transaction is always delivered to the specific process to which it is addressed. FM/360 switches its attention from generic messages to specific messages when it has accepted a "begintool" to process.

2.5.1. G/FM-BEGINTOOL

The "begintool" procedure creates an instance of a tool for a specific Works Manager process (the caller) and a specific Front End process (identified in the transaction). In this implementation, FM/360 can be thought of as basically existing for the purpose of executing the "begintool" procedure. Whenever an instance of FM/360 is active, it is probably either waiting for work or processing as a result of a "begintool" transaction. Most other transactions to which FM/360 responds in other than a trivial way are meaningful only in the context established by a previous "begintool" transaction. Its syntax is:

```
G/FM-BEGINTOOL (program-name,
                tool-type,
                tool-instance-id,
                entry-vector-index,
                Front-End-address,
                creator-address,
                tool-dependent-parameter-list)

          ->   (start-state,
                workspace-descriptor,
                tool-address)
```

If this transaction is completed by an error-mode reply, then no tool instance has been created, and the FM/360 instance sending the reply has dematerialized.

If the transaction is completed normally, then a tool instance exists, and the replying instance of FM/360 is in charge of it and will be receptive to specifically-addressed transactions on its behalf.

2.5.2. G/FM-WMOK

The "WMOK" transaction is intended to let a Foreman know that the NSW Works Manager has crashed and restarted. It is rejected as "unimplemented" by FM/360. The transaction is not parsed, so its syntax is immaterial.

2.5.3. S/FM-ENDTOOL

The "endtool" transaction is intended to terminate execution of a tool, whether or not the tool itself has terminated. The transaction consists only of a single message. The syntax is:

```
S/FM-ENDTOOL (tool-instance-id,
              termination-scenario,
              termination-reason,
              termination-type,
              fault-descriptor)
```

This transaction does not expect a direct reply; however, it and the "toolhalted" transaction are considered to be mutual confirmations of each other.

Due to serious deficiencies in the control-blocking mechanisms of MVT/TSO, if "endtool" is received while the tool task is waiting for terminal input the entire TSO job (Foreman and tool) will deadlock. For this reason, the preferred way to terminate a tool is through the tool's own voluntary termination command.

2.5.4. S/FM-SAVE-LND

The "save-lnd" transaction is intended to suspend execution of a tool, so that it can be resumed later through "rebegintool". Since FM/360 does not support resumption of suspended tools, this transaction is treated like another form of "endtool", with different syntax and a different behavioral scenario. The syntax is:

```
S/FM-SAVE-LND (tool-instance-id,
               termination-reason)

          -> (accounting-list)
```

2.5.5. S/FM-GUARANTEE

The "guarantee" transaction is intended to let a Foreman know that all the information it has entered into the NSW core-system data bases is checkpointed, and can safely be removed from the LND. It is received by FM/360 as a formality, since no actual LND is kept. The syntax is:

S/FM-GUARANTEE (tool-instance-id)

FM/360 will accept and ignore this transaction. At those moments when a more complete Foreman implementation would wait for this transaction before disappearing, so will FM/360; however, it is not waiting to take any action, but only to give the appearance of obeying standard Foreman scenarios.

2.5.6. FOREMAN ALARMS

FM/360 is receptive to three alarms: Alarm code 1 is intended to let a Foreman know that an abort-type "endtool" request is on its way. This allows the Foreman to flush any busy work it is doing on behalf of the tool, and to search forward in its input messages for the promised abort.

A/FM (1)

Alarm code 10 is a request for NSW and Foreman status.

A/FM (10) -> (status-list)

Alarm code 12 is a request for current accounting information.

A/FM (12) -> (accounting-list)

2.5.7. S/FE-OPENCONN

FM/360 issues the "openconn" transaction when it wishes to open the direct connection between the user and the tool. Theoretically, a Foreman can open various connections of various types to the user, including the case of opening none at all; however, FM/360 only supports the case where the tool is controlled by a single TELNET connection. To the tool, this connection appears to be a local terminal operated by a user logged onto TSO. There is no actual reply to this transaction; the Front End responds by actually opening the connection. The syntax is:

```
S/FE-OPENCONN (connection-type,
               process-name,
               connection-id)
```

where "process-name" is always a bitstring of length zero, implying that the sending process is to manage the connection.

2.5.8. S/FE-TOOLHALTED

The "toolhalted" transaction informs the Front End that tool execution is complete, either by voluntary termination or as a result of a previous "endtool" transaction. It is sent to the Front End process named in the "begintool" transaction. When sent to a

Front End, this transaction consists only of a single message; it is considered to be a reply to, or to expect a reply of, the "endtool" message. These two messages are configured in such a way that a race between them will not matter. The syntax is:

```
S/FE-TOOLHALTED (tool-instance-id,
                 termination-scenario,
                 termination-reason,
                 accounting-list,
                 fault-descriptor)
```

where "fault-descriptor" is always a list of count zero.

2.5.9. G/WM-TOOLHALTED

In most particulars, this is just like the "toolhalted" transaction that would go to the Front End. It is only used if an "endtool" transaction cannot be coaxed from the Front End after a reasonable amount of time. Unlike the Front-End variety, this transaction expects a confirming reply; however, no data is returned. The syntax is:

```
G/WM-TOOLHALTED (user-id,
                 tool-instance-id,
                 termination-reason,
                 accounting-list)

             -> ( )
```

where "user-id" is always zero.

2.5.10. G/WM-GET

The "get" transaction makes a tool copy of an NSW file. In theory, this transaction is issued in response to a request from the tool to the Foreman. In FM/360, it is issued only as a result of interpretation of an ECI "GET" statement, which specifies most of its arguments either through calculation, user query, or direct coding. The syntax is:

```
G/WM-GET (attribute-code,
          filespec,
          semaphore-set,
          help-director,
          tool-instance-identifier)

      -> (nsw-filename,
          local-filename,
          new-filespec)
```

Where "help-director" is always zero.

2.5.11. G/WM-DELIVER

The "deliver" transaction makes an NSW file out of a tool copy. In theory, this transaction is issued in response to a request from the tool to the Foreman. In FM/360, it is issued only as a result of interpretation of an ECI "PUT" statement, which specifies most of its arguments either through calculation, user query, or direct coding. The syntax is:

```
G/WM-DELIVER (attribute-code,
              local-file-name,
              entry-name,
              replace-enable,
              help-director,
              tool-instance-identifier)

          -> (nsw-filename)
```

Where "help-director" is always zero.

2.5.12. G/WM-LND-SAVED

The "lnd-saved" transaction informs the Works Manager that execution of one or more tool instances has been terminated abnormally. There are two lists of tools sent to the Works manager. The first is of tools that have been terminated, and the second is of tools that should not be restarted through "rebegintool". The Works Manager returns a list of those tools that it was unaware of. The syntax is:

```
G/WM-LND-SAVED (fm-herald,
                LIST(tool-instance-id),
                LIST(tool-instance-id)

           -> (wm-herald,
               LIST(tool-instance-id)
```

FM/360 sends "lnd-saved" at the end of processing "save-lnd", to inform the Works Manager that the tool just saved should not be restarted. To do this, it includes the single tool instance as the sole element of both lists. In this case, the contents of the returned tool list seem immaterial, so that list is only checked for syntactic correctness.

Figure 4: FM/360 Call Tree.

```
FMDISP
   FMCINIT
   FMSTAT
   FMINIT
   FMREJEC  FMTWAIT  FMANAL *
   FMBEGIN
      FMPCLN
      FMDCRSH    FMFLND **
                 FMCCFE
                 FMDET      TISUBS
      FMDINIT
         FMRBEG      FMTWAIT    FMANAL *
                     FMUID
                     FMREJEC    FMTWAIT    FMANAL *
         FMPBEG      FMREJEC    FMTWAIT    FMANAL *
         FMBLND **
      FMDSAVE
         FMFLND **
         FMMLOST **
         FMMSAVE **
         FMCCFE
         FMDET       TISUBS
         FMWLNDS     FMTWAIT    FMANAL *
         FMRSAVE     FMUID
                     FMTWAIT    FMANAL *
                     FMREJEC    FMTWAIT    FMANAL *
         FMSLND **
         FMFETH      FMDSTAT    FMSTAT
                     FMTWAIT    FMANAL *
      FMDEND
         FMFLND **
         FMCCFE
         FMDET       TISUBS
         FMDELV      ECI
         FMAWGUA     FMANAL *
         FMAWEND     FMANAL *
         FMWMTH      FMTWAIT    FMANAL *
         FMFETH      FMDSTAT    FMSTAT
                     FMTWAIT    FMANAL *
      FMDOPEN
         FMCCFE
         FMOCFE
         ECI
      FMDRUN
         FMRSTRT
         FMANAL *                  ** - These routines are stubbed.
         TISUBS
```

(continued)

Figure 4 (continued): FM/360 Call Tree.

* - "FMANAL" is a recursive subtree:

```
FMANAL      FMALARM     FMDSTAT
                        FMREJEC     FMTWAIT     FMANAL *
            FMSAVE      FMREJEC     FMTWAIT     FMANAL *
            FMEND       FMREJEC     FMTWAIT     FMANAL *
            FMSTOP **
            FMSTART **
            FMREJEC     FMTWAIT     FMANAL *
```

** - These routines are stubbed.

2.6. FM/360 PROGRAM LOGIC

FM/360 has the overall structure shown in figure 4. The following sections detail the functions of the various routines listed there. Some trivial routines are not listed.

ECI -- stands for ECINTRP, the main program of the ECI. Internal logic of the ECI is detailed in the next section.

FMALARM -- processes and replies to all incoming alarms.

FMANAL -- is called whenever any asynchronous event occurs on the PL/PCP (reference 3) communications protocol level. FMANAL functions like a subdispatcher for all the specifically addressed transactions that can be received by FM/360. It analyzes such events and calls appropriate subroutines.

FMAWEND -- awaits an "endtool" transaction when "toolhalted" has been sent first.

FMAWGUA -- awaits a "guarantee" transaction.

FMBEGIN -- Executes the "begintool" transaction. FMBEGIN is basically a state-driven subroutine dispatcher that routes control according to a master state variable with values:

Initializing
Opening
Running
Ending
Saving
Crashing
Done

FMBLND -- is a stub for the routine that will build the LND.

FMCCFE -- closes the TELNET connection to the front end.

FMCINIT -- initializes the FM/360 common data area.

FMDCRSH -- directs control through the "crashing" state, which is entered when FM/360 finds itself in a logical cul-de-sac.

FMDELV -- performs post-tool-execution file delivery. In the present implementation of FM/360, this consists of simply returning control to the ECI to continue interpreting the ECI program that was interrupted to run the tool program.

FMDEND -- directs control through the "ending" state, which is entered when the tool terminates, or when "endtool" is received.

FMDET -- detaches the tool subtask.

FMDINIT -- directs control through the "initializing" state, which is entered when a "begintool" is first executed.

FMDISP -- is a generic transaction dispatcher. It is the main program of FM/360, which awaits a generic transaction, identifies the requested procedure, and calls the appropriate processing subroutine. In the current implementation, it is virtually null, since it can only really call FMBEGIN.

FMDOPEN -- directs control through the "opening" state, which is entered immediately after initialization is complete, and again any time the TELNET connection breaks.

FMDRUN -- directs control through the "running" state. In this state, FM/360 is basically idle, waiting for the tool program to finish, or for a specifically-addressed NSW transaction.

FMDSAVE -- directs control through the "saving" state. In the current implementation, no LND saving actually occurs, but this is because the routines that this one calls are stubbed.

FMDSTAT -- calculates differences in statistical readings taken at various times during tool execution, particularly for status queries and final accounting.

FMEND -- parses and distributes the arguments of "endtool".

FMFETH -- builds a "toolhalted" transaction and attempts to send it to the Front End.

FMFLND -- is a stub for the routine that will free the LND.

FMINIT -- reads the FM/360 configuration data set.

FMMLOST -- is a stub for a routine to handle the case where contact with the Works Manager is lost during "saving" state.

FMMSAVE -- is a stub for a routine to do whatever processing is indicated when "lnd-saved" has been successfully delivered to the Works Manager.

FMOCFE -- opens a TELNET connection to the Front End.

FMPBEG -- parses the "begintool" parameters.

FMPCLN -- cleans up storage allocations supporting the ECI.

FMRBEG -- replies to the "begintool" transaction.

FMREJEC -- rejects a pending transaction with a parametric error code.

FMRSAVE -- replies to the "save-lnd" transaction.

FMRSTRT -- types a message identifying the codes associated with a tool program ABEND. If so instructed by a switch in the FM/360 initialization parameters, allows the user the opportunity to restart the failing tool program or to enter TSO TEST. These capabilities are for FM/360 debugging, and are not normally enabled in the production NSW.

FMSAVE -- processes the "save-lnd" transaction.

FMSLND -- is a stub for a routine to actually save an LND.

FMSTART -- is a stub for a routine to process the "starttool" transaction.

FMSTAT -- reads the various statistical "clocks" that are processed by FMDSTAT.

FMSTOP -- is a stub for a routine to process the "stoptool" transaction.

FMTWAIT -- awaits completion of a given transaction, or arrival of a more important one (such as an "endtool").

FMUID -- locates the TSO "userid" under which FM/360 is operating. The userid defines the "tool workspace" name.

FMWLNDS -- builds and sends an "lnd-saved" transaction.

FMWMTH -- builds a "toolhalted" transaction and attempts to send it to the Works Manager.

TISUBS -- stands for an Assembler-language subroutine package that interfaces with OS/MVT's "task management" facilities. It provides "attach", "detach", "start", and "stop" services.

Figure 5: ECI Call Tree.

```
ECINTRP ECALLOC  NEXTINT   NEXTEL    EVAL *
                                     NEXTBRK
                 NEXTSTR   NEXTEL    EVAL *
                                     NEXTBRK
                 PUTSYMB             TRIMB
                                     UPCASE
                 TRIMB
                 UPCASE
                 PLIDAIR
         ECDELET NEXTSTR   NEXTEL    EVAL *
                                     NEXTBRK
                 PUTSYMB             TRIMB
                                     UPCASE
                 TRIMB
                 PLIDAIR
         ECDUMP  TRIMB
         ECELIF  NEXTEL    EVAL *
                           NEXTBRK
         ECELSE
         ECEND
         ECENDL
         ECEXEC  NEXTSTR   NEXTEL    EVAL *
                                     NEXTBRK
                 PUTSYMB   TRIMB
                           UPCASE
                 TRIMB
                 UPCASE
         ECFREE  NEXTSTR   NEXTEL    EVAL *
                                     NEXTBRK
                 TRIMB
                 UPCASE
                 PLIDAIR
         ECFREED NEXTSTR   NEXTEL    EVAL *
                                     NEXTBRK
                 TRIMB
                 UPCASE
                 PLIDAIR
```

(continued)

Figure 5 (continued): ECI Call Tree.

```
ECGET     NEXTEL    EVAL *
                    NEXTBRK
          NEXTINT   NEXTEL    EVAL *
                              NEXTBRK
          TRIMB
          NEXTSTR   NEXTEL    EVAL *
                              NEXTBRK
          PUTSYMB   TRIMB
                    UPCASE
          SETQ
          UPCASE
ECIF      NEXTEL    EVAL *
                    NEXTBRK
ECINCLU   FRECFM    UPCASE
          FREEWB
          GETWB
          NEXTEL    EVAL *
                    NEXTBRK
          PUTSYMB   TRIMB
                    UPCASE
          SUBST     EVAL *
                    GETSYMB   TRIMB
                              UPCASE
          TRIMB
          UPCASE
          PLIDAIR
ECINDEX   NEXTEL    EVAL *
                    NEXTBRK
          COMPARE   UPCASE
          NEXTLST   NEXTBRK
                    SUBST     EVAL *
                              GETSYMB   TRIMB
                                        UPCASE
          PUTSYMB   TRIMB
                    UPCASE
          SETBRK    GETSYMB   TRIMB
                              UPCASE
          SUBST     EVAL *
                    GETSYMB   TRIMB
                              UPCASE
          TRIMB
          UPCASE
```

(continued)

Figure 5 (continued): ECI Call Tree.

```
ECINTER  FREEWB
         GETWB
         TRIMB
ECMENU   ECINPUT  NEXTEL    EVAL *
                            NEXTBRK
                  SETBRK    GETSYMB   TRIMB
                                      UPCASE
                  TRIMB
         EVAL *
         NEXTEL   EVAL *
                  NEXTBRK
         NEXTLST  NEXTBRK
                  SUBST     EVAL *
                            GETSYMB   TRIMB
                                      UPCASE
         PUTSYM   TRIMB
                  UPCASE
         TRIMB
         UPCASE
ECNOTE   NEXTEL   EVAL *
                  NEXTBRK
ECPUT    NEXTEL   EVAL *
                  NEXTBRK
         NEXTINT  NEXTEL    EVAL *
                            NEXTBRK
         TRIMB
         NEXTSTR  NEXTEL    EVAL *
                            NEXTBRK
         PUTSYMB  TRIMB
                  UPCASE
         SETQ
         UPCASE
ECQUERY  ECINPUT  NEXTEL    EVAL *
                            NEXTBRK
                  SETBRK    GETSYMB   TRIMB
                                      UPCASE
                  TRIMB
         EVAL *
         NEXTEL   EVAL *
                  NEXTBRK
         NEXTLST  NEXTBRK
                  SUBST     EVAL *
                            GETSYMB   TRIMB
                                      UPCASE
```

(continued)

Figure 5 (continued): ECI Call Tree.

```
                PUTSYMB   TRIMB
                          UPCASE
                TRIMB
                UPCASE
ECCREATE        NEXTINT   NEXTEL    EVAL *
                                    NEXTBRK
                NEXTSTR   NEXTEL    EVAL *
                                    NEXTBRK
                PUTSYMB   TRIMB
                          UPCASE
                TRIMB
                UPCASE
                PLIDAIR
ECSET           NEXTEL    EVAL *
                          NEXTBRK
                PUTSYMB   TRIMB
                          UPCASE
ECSETQ          NEXTSTR   NEXTEL    EVAL *
                                    NEXTBRK
                PUTSYMB   TRIMB
                          UPCASE
                SETQ
                TRIMB
                UPCASE
ECSTOP          NEXTEL    EVAL *
                          NEXTBRK
ECSUBSC         NEXTEL    EVAL *
                          NEXTBRK
                NEXTLST   NEXTBRK
                          SUBST     EVAL
                                    GETSYMB   TRIMB
                                              UPCASE
                PUTSYMB   TRIMB
                          UPCASE
                SETBRK    GETSYMB   TRIMB
                                    UPCASE
ECWHILE         FREEWB
                GETSYMB   TRIMB
                          UPCASE
                GETWB
                NEXTEL    EVAL *
                          NEXTBRK
```

(continued)

Figure 5 (continued): ECI Call Tree.

```
        GETSYMB  TRIMB
                 UPCASE
        NEXTBRK
        PUTSYMB  TRIMB
                 UPCASE
        SETBRK   GETSYMB  TRIMB
                          UPCASE
        SUBST    EVAL *
                 GETSYMB  TRIMB
                          UPCASE
        TRIMB
        UPCASE
```

---

* -- EVAL is a common recursive subtree:

```
EVAL    GETSYMB  TRIMB
                 UPCASE
        SUBST    EVAL *
        TRIMB
        UPCASE
```

2.7. ECI PROGRAM LOGIC

The external specifications of the ECI are covered in detail in reference 7, and are not repeated in this document; however, reference 7 does not deal with program logic. The ECI subcomponent has the overall structure shown in figure 5. The functions of its major routines are:

COMPARE -- compares two character strings without regard to case.

ECALLOC -- processes the ECI ALLOCATE statement.

ECDELET -- processes the ECI DELETE statement.

ECDUMP -- dumps ECI storage for debugging.

ECELIF -- processes the ECI ELIF statement.

ECELSE -- processes the ECI ELSE statement.

ECEND -- processes the ECI END statement.

ECENDL -- frees resources allocated to an exiting recursion level.

ECEXEC -- processes the ECI EXECUTE statement.

ECFREE -- processes the ECI FREE statement.

ECFREED -- processes the ECI FREED statement.

ECGET -- processes the ECI GET statement.

ECIF -- processes the ECI IF statement.

ECINCLU -- processes the ECI INCLUDE statement.

ECINDEX -- processes the ECI INDEX statement.

ECINPUT -- gets ECI input from the user teminal.

ECINTER -- processes the ECI INTERPRET statement.

ECINTRP -- is the ECI main program. For any recursion level, it loops through the program, identifying statements and calling the corresponding statement processors.

ECMENU -- processes the ECI MENU statement.

ECNOTE -- processes the ECI NOTE statement.

ECPUT -- processes the ECI PUT statement.

ECQUERY -- processes the ECI QUERY statement.

ECCREATE -- processes the ECI CREATE statement.

ECSET -- processes the ECI SET statement.

ECSETQ -- processes the ECI SETQ statement.

ECSTOP -- processes the ECI STOP statement.

ECSUBSC -- processes the ECI SUBSCRIPT statement.

ECWHILE -- processes the ECI WHILE statement.

EVAL -- is the expression evaluator which processes every operand to every ECI statement.

FRECFM -- determines the record format of a data set.

FREEWB -- frees storage associated with an ECI recursion level

GETSYMB -- looks up the value of a variable symbol.

GETWB -- gets storage associated with an ECI recursion level

NEXTBRK -- scans forward in the ECI statment for a delimiter.

NEXTEL -- breaks out and evaluates the next expression in the statement.

NEXTINT -- breaks out an expression required to be an integer.

NEXTLST -- breaks out the elements of a sublist.

NEXTSTR -- breaks out an expression that is a bounded-length character string.

PUTSYMB -- gives a value to a variable symbol.

SETBRK -- manages alternate values for SYSLDELIM and SYSRDELIM.

SETQ -- determines whether a string is quoted.

SUBST -- recursively substitues values for variable-symbol names.

TRIMB -- trims leading and trailing blanks from character strings.

UPCASE -- converts character strings to upper case.

2.8. APPENDIX: FM/360 INITIALIZATION PARAMETERS

FM/360 decodes a set of initialization parameters from a configuration data set which may optionally be supplied under file name (DDNAME) PARMS. This data set is in the form of a PL/I GET DATA input stream. The following data may be specified, where each name should be qualified by the name "P.":

Name:	Type:	Default:	Meaning:
GENERICNAME	CHAR	'FOREMAN'	FM/360's MSG generic name.
MSG_TIMEOUT	FIXED	60,000	MSG message timeout value, in 0.01 seconds.
PCP_TIMEOUT	FIXED	600,000	PCP transaction timeout value, in 0.01 seconds.
END_TIMEOUT	FIXED	60,000	Timeout value for waiting for "endtool" following "toolhalted", 0.01 seconds.
GUARANTEE_TIMEOUT	FIXED	6,000,000	Timeout value for waiting for "guarantee".
GMT_ADJUSTMENT	FIXED	8.0	Number of hours EARLIER than Greenwich to assume the local clock to be running. The value may be signed (for the Eastern hemisphere) and may carry the fraction ".0" or ".5" (for half-hour time zones).
VOLUME	CHAR	'NSWP01'	Direct-access volume on which to create data sets in the tool work space.
CONFIRM_TERM	BIT	'0'B	Switch to enable debugging options following termination of a tool program.

REFERENCES

1) Schantz and Millstein, "The Foreman: Providing the Program Execution Environment for the National Software Works". Document CADD-7701-0111, Massachusetts Computer Associates, Wakefield, Massachusetts, January 1, 1977.

2) Feinler and Postel (eds.), "Arpanet Protocol Handbook". Document NIC-7104, Network Information Center, SRI International, Menlo Park, California, January, 1978.

3) Ludlam, "PL/PCP -- An NSW Procedure-Call Protocol Package for PL/I". UCLA/OAC document UCNSW-402, November 15, 1980.

4) Ludlam and Rivas, "PL/MSG -- An MSG Interface for PL/I". UCLA/OAC document UCNSW-401, November 15, 1980.

5) Braden, "PLOXI -- A PL/I Interface to Exhange". UCLA/OAC document UCNSW-407, November 15, 1980.

6) Braden, "PL/B8 -- A PL/I Interface Package for NSWB8". UCLA/OAC document UCNSW-403, November 15, 1980.

7) DeLa Roca and Ludlam, "The UCLA Encapsulator Command Interpretor System". UCLA/OAC document UCNSW-206, April 23, 1980.

8) Braden and Ludlam, "FP/360 -- The NSW MVT File Package". UCLA/OAC document UCNSW-204, November 20, 1980.

9) IBM Corporation, "IBM System/360 Operating System Time Sharing Option -- Command Language Reference". Document GC28-6732, 1973.

PART III

The UCLA ENCAPSULATOR COMMAND INTERPRETOR

This section is separately available
as UCLA document UCNSW-206

ILLUSTRATIONS

3. PART III: THE ENCAPSULATOR COMMAND INTERPRETOR

3.1. INTRODUCTION

The UCLA Encapsulator Command Interpretor (ECI) is an interpretor for a primitive language used by NSW Tool installers to specify the interactions between the tool Foreman and the tool user. The ECI language allows the tool installer to query the tool user for information, using menus or simple questions. Using this information, the installer can construct further ECI statements which, when interpreted, can direct the aquisition and delivery of NSW files that are to be used by the tool execution instance.

The effect of the ECI program is to provide a useable interface among the NSW, an NSW user, and an unmodified, non-NSW tool that has been encapsulated within NSW. Because true tool encapsulation, as envisioned by the designers of NSW, will probably never be feasible under an unmodified IBM operating system, the UCLA NSW implementation simulates it by providing comprehensive setup and cleanup of the tool's execution environment. Between setup and cleanup, the tool runs, for all practical purposes, in native mode. The vehicle for specifying this setup and cleanup is the ECI language.

In this document, we use the term "programmer" to refer to the person writing and installing an ECI program. We use the term "user" to refer to the person who invokes the NSW tool with which the ECI program is associated. Thus it is correct to say that the ECI programmer writes a program which communicates with the tool user.

Despite its orientation toward encapsulated tools, the ECI has been installed as an integral part of the UCLA NSW Foreman, and it is available for installing any tool, whether encapsulated or not.

Figure 1 shows a simple and fairly self-explanatory example of an ECI program.

Figure 1. ECI Program Example

```
*************************************************************************
*                                                                       *
*  E C I   P R O G R A M   T O   C O N T R O L   A   T O O L   *
*        T O   C O M P R E S S   A   L I B R A R Y                      *
*                                                                       *
*************************************************************************
SET ABORT , 0
SET CREATERC , 1
WHILE  [CREATERC] ¬= 0
     SET FILESPEC , ' '
     WHILE  [FILESPEC] = ' '
        QUERY QUERYRC , FILESPEC ,                                    -
                  "<filespec><cr> or ?<cr>:  " ,                      -
                  ("Enter the NSW file specification of the -
                        library to be compressed," ,                  -
                        "  or <control-C> to abort.")
        IF  [QUERYRC] < 0
            THEN
               NOTE "Aborting tool session"
               SET CREATERC , 0
               SET ABORT , 1
               SET FILESPEC , '*'
        ELIF  [FILESPEC] ¬= ' '
            THEN
               GET  CREATERC , NSWFILENAME , LOCALNAME , ,    -
                        QNSWCOPY , MEMBER  , [FILESPEC] , 1 , 0
               IF  [CREATERC] ¬= 0
                   THEN
                      NOTE                                            -
                         "unable to locate library [filespec]"
               END
        END
     END
END
IF  [LOCALNAME] ¬= ''
    THEN
       IF [ABORT] = 0
          THEN
             EXEC  RC , COMP [LOCALNAME]
       END
       IF  [QNSWCOPY] = 0
           THEN
              DELETE  , [LOCALNAME]
           ELSE
              FREED  , [LOCALNAME]
       END
END
```

3.2. LANGUAGE OVERVIEW

ECI programs are associated with tools through the optional "Tool Dependent Parameter List" (TPDL) of the Tool Descriptor in the Works Manager's data base. The TDPL may contain an entire ECI program; however, it is more common for it to refer, via an "INCLUDE" statement, to a program stored on the tool's host.

The ECI will interpret the following basic command statements:

ALLOCATE an existing data set to an IBM OS file.
CREATE an empty local data set.
DELETE a local data set.
EXECUTE a TSO command.
FREE a bound IBM OS file name.
FREED a bound local data set name.
GET an NSW file into a local data set.
INCLUDE an ECI program from a local data set.
INDEX of a string within a list.
INTERPRET dynamically entered ECI statements.
MENU display and selection.
NOTE information on the user's terminal.
PUT a local data set into the NSW file space.
QUERY the user for variable information.
SET a symbol equal to value of an expression.
SETQ a flag with by whether a string is quoted.
SETX a symbol with a corrected data set name.
STOP interpretation of an ECI program.
SUBSCRIPT a list.

Plus the following control command statement groups for program structuring:

WHILE / END for iteration.
IF / THEN / ELIF / ELSE / END for selection.
END for program and structure closure.

3.3. LANGUAGE STRUCTURE

As a programming system the ECI language was designed with simplicity and modularity in mind: simplicity in the syntax of the language, and modularity in terms of easy extendability of the language at a later date.

In comparison with traditional generalized programming languages, the ECI language consists of data literals, string variables, string lists, generalized expressions, an "include" facility, and structured programming "select" and "iteration" program structures (there is no GO TO statement or equivalent). The syntax and semantics of all these elements are fully defined and illustrated in appendices of this document. The following sections are only an introductory overview for the uninitiated.

3.3.1. INTERPRETIVE EXECUTION

ECI programs are meant to be interpreted, that is, checked and executed dynamically, one statement at a time. As a rule, the programs are quite short, are written by "expert" tool maintenance personnel, and are stored in and executed from a quality-controlled data base. In such an environment, the drawbacks of interpreting are minimal, and the characteristics of interpretive systems can be exploited to yield a string-oriented programming system which, though primitive, is quite powerful.

The fact that the ECI is not intended for use by the casual programmer has led to an implementation that assumes that all programs are debugged. It is both tolerant and intolerant of errors: most constructs encountered will be interpreted in some way, whether meaningful or not, but those which cannot be interpreted at all will cause program interpretation to be aborted without comment.

3.3.2. DATA TOKENS

At its most basic level the ECI language operates on character data tokens. No other data token is defined in the language. Any literal string of arbitrary characters is an ECI data token. Matched quotes, either single or double, can be used to cause blanks to be a part of such a string. Internal quotes are represented via the usual doubling convention, so that the value of a character data token is the literal string itself, minus possible surrounding quotes (of either type, so long as they match), and with all enclosed dual occurrences of quotes (of the type bracketing the string) collapsed to single occurrences.

In those contexts requiring arithmetic values, strings are coerced to arithmetic values, with the value "0" being used as a last resort. Where a boolean value is required, the arithmetic value "0" becomes the boolean value FALSE, and any other arithmetic value becomes TRUE.

There are some rare cases where an ECI statement behaves differently when given numeric input than when given a non-numeric string (see the "<gft or attrno> argument of the GET and PUT statements as examples). In order to talk about such cases, the ECI defines the "numeric validity" and the "boolean validity" of a string as TRUE when the string requires no coercion to represent a valid numeric or boolean value.

In normal programming usage the above distinctions are of no concern to the programmer; most of the time the right thing happens. However, the ECI programmer should always be aware that the internal representation of all ECI data values is the character string. Due to the coercion conventions the ECI interpreter will usually run happily even when given the most absurd sorts of expressions. It is assumed that the programmer will limit himself to vaguely meaningful statements.

3.3.3. VARIABLE SYMBOLS

ECI variables are implemented as a general character substitution mechanism, much like those of macro assembler languages. The construct

[string]

denotes the character value of a variable whose name is given by the literal string enclosed in the brackets. This means that the ECI interpreter implements a symbol table in which both name and value spaces are dynamically assignable by the programmer. Any string may appear within brackets -- no explicit declaration is required -- but its character value will be a null string until something else is explicitly assigned it. Arbitrary nesting of brackets is allowed, and the clever programmer can use this feature to advantage (see the section entitled "ARRAYS").

Variable symbols may occur anywhere within an ECI statement. Square brackets are permanently reserved for this purpose, and are therefore ferreted out in any context, even within quoted strings, to trigger symbol table substitution. Brackets are rarely used in IBM systems, so that reserving them for this purpose is not usually an inconvenience. Should it be necessary for your ECI program to treat brackets as data characters, see the definitions of SYSLDELIM and SYSRDELIM in the section entitled "System Variable Symbols."

A string within brackets is considered to be free of the effects of alphabetic case, so that "[alpha]", "[ALPHA]", and "[Alpha]" are all the same variable symbol. Of course, case is always preserved within the VALUE of a variable symbol.

3.3.4. SYSTEM VARIABLE SYMBOLS

Certain symbol table variables, designated "system variable symbols," have non-null initial values. All these symbols have names beginning with "SYS" and their values represent both program and environment parameters. Some of these variables are meant to be read-only, even though such usage is not enforced by the interpreter, while the other variables are intended to be set by the programmer to specify ECI trace modes, etc. The system variable symbols currently defined are:

SYSLDELIM is a read-only system variable yielding as its value the left bracket symbol used to denote symbol table value substitution. A character introduced into the ECI program through this variable symbol will NOT trigger variable symbol substitution.

SYSRDELIM is a read-only system variable yielding as its value the right bracket symbol used to denote symbol table value substitution. A character introduced into the ECI program through this variable symbol will NOT trigger variable symbol substitution.

SYSDATE is a read-only system variable yielding as its value the date in the form "mm/dd/yy". This represents the date that processing of the ECI main program began, and it does not vary during the execution of an ECI program and its subprograms.

SYSDATEX is a read-only system variable yielding as its value the same date as SYSDATE, but in the form "mmddyy".

SYSTIME is a read-only system variable yielding as its value the time in the form "hh/mm/ss". This represents the time that processing of the ECI main program began, and it does not vary during the execution of an ECI program and its subprograms.

SYSTIMEX is a read-only system variable yielding as its value the same time as SYSTIME, but in the form "hhmmss".

SYSACCT is a read-only system variable yielding as its value the account name under which the ECI is running.

SYSUID is a read-only system variable yielding as its value the TSO "userid" under which the ECI is running.

SYSACCT is a read-only system variable yielding as its value the directory name under which the ECI is running. This the prefix that is assumed for non-quoted local data set names.

SYSCNT is a read/write system variable yielding as its value the current watchdog loop iteration limit. No iteration control structure is allowed to iterate beyond this limit. Its initial value is 100.

SYSECHO is a read/write system variable yielding as its value the current setting of the "echo trace" flag: "ON" for enabled, "OFF" for disabled. Its initial value is "OFF". When enabled, the echo trace produces a trace, on the currently defined journal file, of every ECI statement actually executed, with all symbol substitutions shown. The trace is a useful means to debug programs.

SYSTRACE is a read/write system variable yielding as its value the current value of the "system trace flag": "ON" for enabled, "OFF" for disabled. Its initial value is "OFF". When enabled, the system trace produces a trace, on the currently defined journal file, of every symbol table change, of every statement executed, and of the interpretor control blocks at each time of execution. This trace is intended to support ECI maintenance. It should be used by programmers when suspecting problems with the interpreter itself.

3.3.5. EXPRESSIONS

By a specific design decision, every parameter of every ECI statement is a generalized expression which is evaluated every time the interpreter executes the statement.

Expressions in the ECI language are implemented in the most general sense. The usual arithmetic, relational, and logical operators are supported with their traditional evaluating priorities. As usual too, parentheses may be used to specify associativity. ECI expressions operate on character data tokens, coerced to arithmetic or boolean values when necessary. Symbol table value substitution is done in such a way that the original syntax of the expression is never altered. The result of expression evaluation is converted to a character string and substituted into the original ECI statement.

The ECI programmer should remember that the arithmetic coercion conventions of the ECI make almost any expression legitimate. For instance,

```
APPLES + 1ORANGE
```

is legitimate and has the value 0+1 = 1.

3.3.6. STRING MANIPULATION

There are no operators for breaking strings apart, since the specialized ECI application does not yet require them; however, statements implementing substring and scan operations can easily be added.

Concatenation, on the other hand, is a needed capability, and is supported implicitly, rather than through an explicit operator. Runs of quoted and non-quoted strings, with or without intervening blanks, yield in effect a single data token. In the evaluation of expressions, two neighboring expressions with no recognizable operator between them are evaluated separately and their character results then concatenated.

3.3.7. LISTS

An important construct in the ECI language is the list structure, consisting of a sequence of expressions or literal strings separated by commas, and sometimes bracketed by parentheses. The comma is another reserved symbol in the ECI language; however, unlike brackets, commas may be hidden within quotes or parentheses. The construct

```
'a,b',c
```

denotes a two element list separated by a level-0 comma, and

```
(a,b),c
```

is a list consisting of one sublist element and an atomic element.

The rules for breaking out items of a sublist are the same as for a top-level list. Leading and trailing blanks are not significant unless within quotes, and the outer level of quotes is not a part of the item's value. The parentheses surrounding a sublist are discarded.

Lists are primarily used as arguments to ECI statements. The only mechanisms for manipulating lists are the INDEX and SUBSCRIPT statements.

3.3.8. ARRAYS

Arrays are not explicitly supported by the ECI language; however, they are easily simulated by a clever programmer. Because arbitrary nesting of brackets is allowed, strings such as

[VECTOR[N]]

[MATRIX[N]X[M]]

can be set and used. Well nested variable symbols are substituted from the inside out. This technique could be used to build more complex data aggregates.

3.3.9. STATEMENTS

ECI program statement structure is utter simplicity: each statement consists of a command field or verb and a list of zero or more parameters separated by level-0 commas. Any parameter may be a sublist, in which case it must be delimited by parentheses.

The basic cycle of the interpreter consists in stepping thru each of the ECI statements, invoking the named command processor, and passing to it the unevaluated parameter list as an argument. The command processor evaluates each parameter expression, as needed to perform its task, and returns control to the interpreter. For the "control statements," the interpreter also keeps status and control information that governs nonsequential flow of control.

3.3.10. PROGRAMS

Program structures are also quite simple. There are three such structures: programs, select structures, and iteration structures.

A program is nothing more than a collection of ECI statements delimited in some way external to the ECI syntax. The TDPL datum of the FM-BEGINTOOL procedure call is such a program. Other programs are local data sets or members of local libraries. A special form of program can be entered from the tool user's keyboard in response to an INTERPRET command. The external representation of programs is defined in the section entitled EXTERNAL ECI REPRESENTATIONS.

One program may invoke another via the INCLUDE or INTERPRET statements. Since the second program is run under the symbol table space of the first program, these mechanisms constitute a primitive subroutine capability, with communication between programs being possible via shared symbol table variables. However, these mechanisms may be time-consuming, so they should not be used carelessly in heavily iterated parts of a program. An ECI program that was not invoked via INCLUDE or INTERPRET is called an ECI main program; all others are called subprograms. All programs stored in TDPL's are main programs.

The select control structure specifies, in its simplest form, the usual "IF" conditional construct, and in its most general form, the "CASE" construct of other languages. Similarly, the iteration control structure emulates the WHILE construct of other languages.

Both control structures are explicitly closed by an END statement. Notice that while these control structures span multiple ECI statements, the ECI interpreter operates on only one statement at a time, and is not sensitive to anomalous groupings of statements. Nonsensical structures may generate unexpected results, including abnormal termination of the ECI program. As already mentioned, the interpretor is rather permissive and relies on normal usage by an expert programmer.

3.4. EXTERNAL ECI REPRESENTATIONS

There are three ways to enter an ECI program into NSW: through the Tool dependent Parameter List of the Tool Descriptor in the Works Manager Data Base; through a local data set to be read via the INCLUDE statement; and directly into the NSW Front End when an INTERPRET statement is processed. The statement syntax is the same in all cases, but the representation on each entry medium differs according to the need for a statement continuation convention.

The TDPL consists of a list of character strings of arbitrary length; therefore, it is not necessary to continue ECI commands stored there. Any continuation convention is thus defined by the program that creates the Tool Descriptor. Works Manager Tool Descriptors are created by utilities supplied and documented by the Works Manager developers. The reader is referred to Works Manager documentation for more information on using these tools.

A local data set to contain ECI commands can be sequential or partitioned, and can have any IBM OS record format that does not include "carriage control" codes. The number of data characters per line is limited to 255, so the OS LRECL must not exceed 259 for format-V records, or 255 for other formats. For line printer compatibility, it is usually desirable to use a much shorter line length, so a mapping of input records into ECI statement lines is defined. This mapping is independent of the ECI statement syntax.

* An input line contains a "significant part" consisting of the line less all leading and trailing blanks. A line with a null significant part contributes nothing to the text of an ECI command.

* In the simple case, an ECI statement consists of the significant part of an input line. However, if that results in a statement terminated by a hyphen, "-", a continued statement is indicated.

* Whenever a significant part is found to terminate with a hyphen, the hyphen is deleted, and the significant part of the following input line is concatenated in its place. This continuation process is applied recursively until either: 1) a line is found with a significant part not terminating with a hyphen (this includes a null line); or 2) the end of the data set is reached (the final hyphen is deleted anyway).

* Notice that blanks before a hyphen are part of the significant part of a line, but that leading blanks of the continuation line are not. Notice too, that a significant part, once having had a hyphen deleted, is never checked for a second hyphen. In a case where a statement absolutely must terminate with a significant hyphen, use two hyphens followed by a null line.

When entering an ECI statement through an NSW Front End, the statement is terminated by a keyed carriage return. All Front Ends as of this writing are full duplex, and a long string is automatically placed on successive lines of the terminal display medium, without the need to key carriage returns. Continution characters are not defined in this case, and should not be used. Notice that this form of statement entry is intended for tool development personnel, and will be used infrequently.

3.5. IMPLEMENTATION LIMITS

The primary objectives in the design of the ECI language were simplicity and ease of use by experts. The design of the ECI run time environment stressed simplicity and minimal error checking. A number of automatic data coercions and defaults were implemented to improve error tolerance, and there is minimal error checking. Consequently most ECI programs will execute when most other languages would give up.

A few physical limits must be observed, but they will be of consequence only in the most extreme cases. Terminal input, statement parameters, and expression results will be truncated at 256 characters. ECI program lines and ECI lists will be truncated at 1024 characters. Truncation never results in any error message.

There are no built-in limits on the maximun number of parameters per ECI statement, size of program units, maximum nesting of control structures, etc. These are bounded by the amount of storage available to the ECI -- the program runs until its internal resources are exhausted and then gives up ungracefully.

The majority of the ECI's dynamic storage is used to contain the ECI programs and the symbol table. TDPL programs are stored as they are received. Included programs are stored as tightly packed as is practical. The symbol table is primitively implemented, and lacks any garbage collection mechanism; however, it is not expected that symbol table activity will usually be great enough to require one.

3.6. ECI STATEMENT DEFINITIONS

This section consists of an alphabetic list of the ECI statements. For each one, the parameter syntax is shown and the meaning of each parameter is explained. By convention, any outputs produced by a statement are the first parameters, with input arguments following. Note that when a parameter is described as "the name of a variable symbol" that name is NOT enclosed in brackets. A name in brackets represents the VALUE of a variable symbol, not its name.

Data set names and patterns can be quoted or non-quoted; however, a quoted literal string should be further enclosed in double quotes to ensure that the single quotes remain a part of the parameter value.

3.6.1. ALLOCATE STATEMENT

The ALLOCATE elementary statement binds an existing data set to an IBM OS file name (DDNAME). The syntax is:

```
ALLOCATE <retcode>,<ddname>,<dsname>,<member name>
     <share flag>,<check>
```

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

```
 0 --> successful allocation
 8 --> local data set can't be located
12 --> insufficient resources
16 --> any other error
```

<ddname> is the IBM OS "DDNAME"; that is, the name of the file to be bound.

<dsname> is the local name of the data set to be bound to the file.

<member name> is the name of a member of the data set, if it is a library, and if and only a member is to be bound. In other cases, it is null or blank.

<share flag> is a boolean. It is 1 if the allocation can be shared with other concurrent shared users, and 0 if the allocation must be for exclusive control.

<check> is a boolean. If it is 1, and if allocation is to a member name, then allocation will not succeed unless the member already exists. If it is 0, successful allocation of a library member only requires that the library exist (this is sufficient if the member is about to be created).

3.6.2. CREATE STATEMENT

The elementary statement CREATE builds a new data set, usually in the local workspace, and remembers its name. The attributes of the data set are derived from a specified Global File Type name. The syntax is:

```
CREATE <retcode>,<dsname>,<dsname pattern>,<gft>,
    <filesize>,<growth>,<directory size>
```

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

```
 0 --> successful execution
 4 --> local data set name already exists
12 --> insufficient resources
16 --> any other error
```

<dsname> is the name of a variable symbol to be assigned the value of the local data set name.

<dsname pattern> is the local data set name to be supplied, except that it may contain up to 7 occurrences of the "wild character" "?". This character will be replaced by any alphanumeric character that will result in a unique name. Note that it is not a pure alphabetic character, so it should not be used as the first character, or immediately following a ".".

<gft> is an NSW Global File Type (GFT) name that represents the data set attributes. It must be native to the 360 family; that is, it must begin with "360-".

<filesize> is an integer stating the initial data set size in kilobits. If it is not present, a default associated with the GFT will be used.

<growth> is an integer stating the data set's growth increment in kilobits. If it is not present, a default associated with the GFT will be used.

<directory size> is an integer stating the number of directory blocks to be reserved for a library. If it is not present, the data set will be sequential. If it is present, the data set will be a library.

3.6.3. DELETE STATEMENT

The elementary statement DELETE simulates the Foreman's DELETE LOCAL primitive by destroying a data set, usually one created by the CREATE statement. The syntax is:

DELETE <retcode>,<dsname>

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

0 --> successful deletion
8 --> unable to locate data set
12 --> insufficient resources
16 --> any other error

<dsname> is the local data set name to be deleted.

3.6.4. ELIF STATEMENT

The control statement ELIF opens up a conditionally executable group within a select control structure. The syntax is:

ELIF <expression>

<expression> has the same function as the same parameter of the IF statement, to which this statement must be subordinate. It determines whether the ELIF group is to be executed. However, an ELIF group is never executed if any previous THEN, ELIF, or ELSE group has already been executed within the containing control structure. If the ELIF group is executed, then the flow of control is directed to the statement immediately following the END statement of the select control structure.

3.6.5. ELSE STATEMENT

The control statement ELSE opens up an unconditionally executable group within a select control structure. The syntax is:

```
ELSE
```

An ELSE group is not executed if any previous THEN, ELIF, or ELSE group has already been executed within the containing control structure. If the ELSE group is executed, then the flow of control proceeds to the statement immediately following the END statement of the select control structure.

3.6.6. END STATEMENT

The END control statement closes program structures. The syntax is:

```
END
```

Both select and iteration control structures are closed by END's matching the corresponding IF's and WHILE's opening those structures.

Programs are normally bracketed by their physical boundaries, as by beginning and end of file; however, a program introduced by the INTERPRET statement has no such physical boundaries, since it is extracted from the user's terminal input stream. In this case, that program is closed by an END statement which can be considered to match an implicit opening delimiter which is the position of the terminal input stream at the point of execution of the INTERPRET statement. For uniformity, the ECI allows the closing of any program with an END statement, whether it is from the terminal stream or not.

It is an ECI requirement that control structures be well nested. Thus a select or iteration structure cannot be opened in a subprogram and closed in its calling program. For this reason, when a program ends with unclosed contained control structures, the ECI will supply enough implicit END statements to make the program well-formed.

3.6.7. EXECUTE STATEMENT

The elementary statement EXECUTE invokes a given program as a TSO command processor. Normally, this will be the tool program itself; however, the command may be used any number of times, and may call any TSO number of TSO command processors. The syntax is:

EXECUTE <retcode>,<command>

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

0 --> program terminated under its own control
12 --> program terminated abnormally

<command> is any character string. Unless it is null, it should be formatted as a TSO command; that is, it should begin with a TSO command verb delimited by blanks and/or the field boundaries. The command is executed exactly as though it were entered outside the NSW system. Its terminal interactions will be directed to the NSW Front End, and will not be available to the ECI.

3.6.8. FREE STATEMENT

The elementary statement FREE unbinds an existing IBM OS file name (DDNAME) from any local data set name to which it is currently bound. The syntax is:

```
FREE <retcode>,<ddname>
```

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

```
 0 --> successful unbinding
 8 --> unable to locate ddname
12 --> insufficient resources
16 --> any other error
```

<ddname> is the IBM OS file name (DDNAME) to be FREE'd.

3.6.9. FREED STATEMENT

The elementary statement FREED unbinds an existing data set from any IBM OS file names (DDNAME's) to which it is currently bound. The syntax is:

```
FREED <retcode>,<dsname>
```

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

```
 0 --> successful unbinding
 8 --> unable to locate dsname
12 --> insufficient resources
16 --> any other error
```

<dsname> is a local data set name to be FREED'd.

3.6.10. GET STATEMENT

The elementary statement GET simulates the Foreman's GET primitive by calling WM-GET. It transforms an NSW filename into a local data set name and remembers it. The syntax is:

```
GET <retcode>,<nsw file name>,<dsname>,
    <altered filespec>,<qnswcopy>,<member name>,
    <filespec>,<gft or attrno>,<qset>
```

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

0 --> successful execution
4 --> file was not gotten

<nsw file name> is the name of a variable symbol to receive the resolved NSW file name.

<dsname> is the name of a variable symbol to be assigned the local data set name.

<altered filespec> is the name of a variable symbol to be assigned the filespec as altered by the user in any HELP calls. If no filespec alteration is done, this variable is assigned a null value.

<qnswcopy> is the name of a variable symbol to be assigned the value "0" if a tool copy is made, or the value "1" if an NSW copy was allocated directly.

<member name> is the name of a variable symbol to be assigned the name of a member, if the GET resulted in the allocation of a member of an NSW library file. Otherwise, this variable is set to a null value. (At this writing, NSW does not yet support member allocation, and this variable is always set to null.)

<filespec> is the NSW "filespec", an abbreviated form of the name of the existing NSW file to be accessed.

<gft or attrno> is interpreted in either of two ways. If its numeric validity is TRUE, then it is a small integer designating an index into the Works Manager's Tool Descriptor's file type vector. When so applied, it yields the Global File Type (GFT) list describing the desired file copy. But if its numeric validity is FALSE, then it is a character string representing a single GFT directly, and so it must begin with "360-". (At this writing, only integers are supported.)

<qset> specifies whether the Works Manager is to set the file semaphore on the NSW file being gotten. Boolean value TRUE requests such setting, while boolean value FALSE requests that it not be set.

3.6.11. IF STATEMENT

The control statement IF opens up a "select control structure" consisting of one or more conditional (THEN, ELIF) and/or unconditional (ELSE) select control groups, and one closure bracket (END). The interpreter implements minimal program structure checking and thus anomalies in the order of these control statements may cause unpredictable results. The expected variations of the select control structure are illustrated in figure 2 (Note that THEN is a null statement, which does nothing but improve the readability of ECI programs). Within each select group, other control structures may be nested to a level limited only by the main storage available to the ECI interpreter. The syntax is:

IF <expression>

<expression> is an arbitrary expression whose boolean value, if TRUE, triggers execution of the control structure THEN group. If the value is false, flow of control proceeds to the next ELIF, ELSE, or END statement.

The general logic for flow of control within a select control tructure is as follows:

1. If the IF expression yields a boolean value TRUE then the group immediately following the IF statement is executed, and then control skips to the statement following the END statement.

2. If the IF expression yields a boolean value FALSE then control skips to the next ELIF, ELSE, or END statement that is a part of this select control structure.

3. If control reaches an ELIF statement, then its expression is evaluated. If it yields a boolean value TRUE, then the ELIF group is executed, and control skips to the statement following the END statement.

4. If control reaches an ELSE statement, then the ELSE group is executed unconditionaly, and control skips to the statement following the END statement.

5. If all conditional expressions on the IF and on any ELIF statements are false, and if there is no ELSE statement, then control will proceed past the END statement with no statements in the control structure having been executed.

6. When control reaches the END statement, whether by skipping to that point or by executing the last statement group in the structure, control proceeds by executing the statement following the END statement.

Figure 2. Expected Forms of Select Control Structures.

```
IF <expression>
   THEN
      <THEN statement group>
END


IF <expression>
   THEN
      <THEN statement group>
   ELSE
      <ELSE statement group>
END


IF <expression>
   THEN
      <THEN statement group>
ELIF <expression>
   THEN                                --| n
      <ELIF statement group>           --| units
   ELSE
      <ELSE statement group>
END


IF <expression>
   THEN
      <THEN statement group>
ELIF <expression>
   THEN                                --| n
      <ELIF statement group>           --| units
END
```

3.6.12. INCLUDE STATEMENT

The elementary statement INCLUDE causes the ECI interpreter to retrieve and execute an ECI program from a local dataset. The included program is logically terminated by an end-of-file or by an unmatched END statement. Execution of the INCLUDE statement does not complete until the entire included program has completed. Both programs use the same ECI variable symbol table space so values may be passed in either direction via variable symbols. If the included program is aborted, execution of the including program is not affected directly; however, the termination status of the included program is reported back to the includer. The syntax is:

INCLUDE <retcode>,<filename>,<member name>

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

0 --> successful execution of program
8 --> program was aborted during execution
16 --> program could not be located
24 --> any other error

<filename> is an IBM OS DDNAME to which the data set is already allocated. If it is null, standard filename "ECILIB" will be used.

<filename> is the name of a member to be retrieved from the library. If it is null, then <file> must be given, and must be allocated to a sequential data set.

3.6.13. INDEX STATEMENT

The elementary statement INDEX searches a list of items for a match on an argument string, and returns a small integer (zero origin) indicating which element was matched. Comparison is without regard to alphabetic case. The syntax is:

INDEX <index>,<argument>,<list>

<index> is the name of a variable symbol to receive the returned integer. The list is considered to be zero origin. A value of -1 is returned if no match is found.

<argument> is a character string to be matched against the list parameter.

<list> is a parenthesized list of potential matches on the value of <argument>.

3.6.14. INTERPRET STATEMENT

The elementary statement INTERPRET is intended for use by tool development personnel in debugging tool *implementations*. It allows dynamic specification of an entire ECI statement sequence; however, this sequence is interpreted as it is entered, and it is not saved. Therefore, execution of all the control statements IF, WHILE, THEN, ELSE, and ELIF is disabled. The syntax is:

INTERPRET

On recognizing this command, the ECI interpreter will switch flow of control from the current executing ECI program to the user's terminal. It will prompt for, accept and execute elementary statements entered at the user's terminal until an END statement or an attention interruption is encountered. Then flow of control is switched back to the executing program at the statement immediately following the INTERPRET statement just executed.

Notice that the ECI's intolerance for errors is not mitigated because the statements entered due to INTERPRET are from the user's terminal. However, an error in a user-entered statement will not abort the ECI program, since the user will have the chance to reenter the statement correctly.

3.6.15. MENU STATEMENT

The elementary statement MENU requests input from the user, and accepts from him a value from a given menu. Help information can be provided. The statement returns both the input string and a small integer designating its zero-origin index in the menu. The syntax is:

MENU <index>,<input>,<prompt>,<menu>,<help>

<index> is the name of a variable symbol to receive the returned integer. The list is considered to be zero origin, and a value of -1 is returned if the user aborts the statement via an attention without providing a match.

<input> is the name of a variable symbol to receive the string input by the user. Notice that it may not be an exact match of any menu item as abbreviations, unambiguous or not, are accepted, and comparison is without regard to alphabetic case. The matching algorithm works very simply -- it stops searching at the first successful match -- therefore the order of the items in the menu list is important if ambiguous abbreviations are possible.

<prompt> is a character string written to the user's terminal to prompt him for input. Normally, this string should include or imply the menu values.

<menu> is a parenthesized list of potential matches for the user's input string. This list is for internal matching by the MENU processor, and should not be formatted for the user.

<help> is a parenthesized list of lines of HELP output. The entire list is written to the terminal if the user enters a "?" in response to the first prompt.

3.6.16. NOTE STATEMENT

The elementary statement NOTE merely types information on the user's terminal. The syntax is:

NOTE <text>

<text> is a character string to be ouput on the user's terminal.

3.6.17. PUT STATEMENT

The elementary statement PUT simulates the Foreman's PUT primitive by calling WM-DELIVER and remembering the resulting NSW file name. The syntax is:

```
PUT <retcode>,<nsw file name>,<new member name>,
    <new member version>,<dsname>,<entry name>,
    <gft or attrno>,<qreplace>
```

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

```
 0 --> successful delivery
 4 --> Works manager won't accept data set
 8 --> Local data set can't be located
12 --> Insufficient resources
16 --> Any other error
```

<nsw file name> is the name of a variable symbol to be assigned the value of the NSW file name.

<new member name> is the name of a variable symbol to be assigned the member name as altered by disambiguation. (At this writing, NSW does not yet support member names, and this string will always be null.)

<new member version> is the name of a variable symbol to be assigned the version number of a member in a library. (At this writing, NSW does not yet support member names, and this string will always be null.)

<dsname> is the local name of the data set to be delivered.

<entry name> is the NSW "entry name", an abbreviated form of the name of the new NSW file to be created.

<gft or attrno> is interpreted in one of two ways. If its numeric validity is TRUE, then it is a small integer designating an index into the Works Manager's Tool Descriptor's file type vector. When so applied, it yields the Global File Type (GFT) describing the file being delivered. If the numeric validity of the expression value is FALSE, then the value represents a character string which is a GFT directly, so it must begin with "360-". (At this writing, only integers are supported.)

<qreplace> specifies whether the Works Manager is to replace any existing NSW file of the same name. 1 requests replacement, and 0 requests further disambiguation.

3.6.18. QUERY STATEMENT

The elementary statement QUERY requests input from the user, and saves his reply. Help information can be provided. The syntax is:

QUERY <retcode>,<input>,<prompt>,<help>

<retcode> is the name of the variable symbol to receive a return code summarizing the results of statement execution. Defined values are:

0 --> a reply was returned
4 --> user aborted with attention

<input> is the name of a variable symbol to be assigned the character string input by the user.

<prompt> is a string message with which the user is to be prompted for input.

<help> is a parenthesized list of lines of HELP output. The entire list is written to the terminal if the user enters a "?" in response to the first prompt.

3.6.19. SET STATEMENT

The elementary statement SET assigns a literal or calculated value to a variable symbol. The syntax is:

SET <symbol>,<value expression>

<symbol> is the name of a variable symbol to receive the value of the parameter expression.

<value expression> is an arbitrary expression whose value is assigned to the given ECI variable.

3.6.20. SETQ STATEMENT

The elementary statement SETQ returns a boolean TRUE if its argument is a quoted string, or a FALSE if it is not. The syntax is:

SETQ <symbol>,<test string>

<symbol> is the name of a variable symbol to be set to 1 if <test string> is a properly quoted string, and to 0 otherwise.

<test string> is a character string to be tested as to whether it is surrounded by matching quote marks (either single or double).

3.6.21. SETX STATEMENT

The elementary statement SETX converts a string containing a quoted partitioned data set name from the form in which it is likely to be generated by string concatenation to that which TSO requires. If the input string is not of the form expected, the output will be an unchanged string. For example, the string:

'alpha.beta'(member)

must be converted to

'alpha.beta(member)'

before it can be processed. The syntax is:

SETX <symbol>,<dsname>

<symbol> is the name of a variable symbol to receive as its value the quoted partitioned data set name in corrected form. If no correction takes place then it is set to the same value as <dsname>.

<dsname> is the data set name to be checked and converted.

3.6.22. STOP STATEMENT

The elementary statement STOP halts execution of the ECI program. Notice that STOP is not usually used for normal program termination, since program execution terminates normally at the logical end of the program. The syntax is:

STOP <abort expression>

<abort expression> indicates whether the STOP is to be considered normal or abnormal for the purposes of reporting back to any INCLUDE command which may be executing this ECI program. If the boolean value of <abort expression> is TRUE, the STOP is considered an abort. Otherwise, it is considered a normal termination.

3.6.23. SUBSCRIPT STATEMENT

The elementary statement SUBSCRIPT is the inverse of INDEX. It accepts a list and a small integer, and returns the list element corresponding to the integer. List indexing is zero-origin. The syntax is:

SUBSCRIPT <symbol>,<index>,<list>

<symbol> is the name of a variable symbol to receive the selected string. If the index is out of range, it will be set to a null string.

<index> is the index to be applied to the list.

<list> is a parenthesized list of items from which the selection is to take place.

3.6.24. THEN STATEMENT

The THEN control statement is a noise statement introduced only to make ECI programs more readable. It is intended to open the group following an IF statement. Depending on your indentation convention, you may also want to use THEN after an ELIF statement.

For example:

```
IF <expression>
   THEN
      <statement group>
   ELSE
      <statement group>
END
```

Rather than:

```
IF <expression>
   <statement group>
ELSE
   <statement group>
END
```

3.6.25. WHILE STATEMENT

The WHILE control statement introduces an iterative control structure which includes all following statements through a matching END statement. Well formed control structures can be nested within a WHILE structure to a depth bounded only by the main storage available to the ECI interpretor. The syntax is:

WHILE <expression>

<expression> is an arbitrary expression whose boolean value, while TRUE, triggers execution of the entire control structure. If its value is initially FALSE, flow of control skips directly to the statement following the matching END statement. If it is TRUE, the statements of the structure are executed, and when the matching END statement is encountered, control returns to the WHILE statement. This process continues until <expression> evaluates to FALSE or until the number of iterations exceeds the value of the watchdog counter SYSCNT. See the section entitled SYSTEM VARIABLE SYMBOLS for a discussion of SYSCNT.

3.6.26. COMMENT STATEMENT

The comment statement is a no-operation statement which allows program annotation. The syntax is:

* <any string of characters>

3.7. APPENDIX A -- ECI LANGUAGE GRAMMARS

All of the grammar descriptions that follow could be presented in pure BNF form; however, the following extensions eliminate recursive notation and aid readability and parser implementation.

< > Angle brackets are used to define nonterminal symbols corresponding to production rules. They may be nested, so that <<ident><cntl>> denotes a nonterminal symbol different from <ident> or <cntl> alone.

::= This compound symbol denotes a grammar production or rewrite rule.

| Denotes an alternation operation.

blank or null: indicates concatenation. Concatenation has precedence over alternation, so that <x>::=<y><z>|<y> means rewrite <x> as either <y><z> or simply as <y>.

{ } Braces are used to indicate zero or more occurrences of the enclosed productions.

[ ] Square brackets are used to indicate an optional production.

( ) Parentheses are used as grouping brackets. Thus <x>::=<y><z>|<y> is not the same as <x>::=<y>(<z>|<y>). Grouping is useful to factor out common parts of production rules.

CAPS or " ": terminal symbols are in all capitals, or are enclosed in double quotes. This makes the double quote a reserved symbol in this notation.

3.7.1. STATEMENT GRAMMAR

3.7.1.1. SYNTAX

<any ECI statement> ::= < blanks> <command verb> <blanks>
[<parameter list>] <eol>

<command verb> ::= ALLOCATE | CREATE | DELETE | ELSE | ELIF |
END | EXECUTE | FREE | FREED | GET | IF |
INCLUDE | INDEX | INTERPRET | MENU | NOTE |
PUT | QUERY | SET | SETQ | SETX | STOP |
STOP | SUBSCRIPT | TASKLIB | THEN | WHILE

<blanks> ::= <one or more blanks>

<parameter list> ::= [<expression> {<separator> <expression>}]

<expression> ::= <a parameter expression as described fully
under "EXPRESSION GRAMMAR">

<separator> ::= [<blanks>] <comma> [<blanks>]

<eol> ::= <end of record> (one statement per record allowed)

3.7.1.2. SEMANTICS

Only level-0 commas function as separators. Any comma enclosed in quotes or within matched parentheses is not a separator. There are two actual parameters in the string:

```
"ASM 10,20,40",(CO 10,15,70)
```

Missing or null parameters are indicated by succeeding commas and optional blanks. There are 3 null parameters and two non-null parameters in the string:

```
, .,A,B
```

3.7.2. PROGRAM GRAMMAR

3.7.2.1. SYNTAX

```
<simple group> ::= {<allocate stmt> | <create stmt> |
                    <delete stmt> | <execute stmt> |
                    <free stmt> | <freed stmt> |
                    <get stmt> | include stmt> |
                    <menu stmt> | <note stmt> |
                    <put stmt> | <query stmt> |
                    <set stmt> | <setq stmt> |
                    <setx stmt> | <stop stmt> |
                    <subscript stmt> | <interpret stmt>}

<while group> ::= <while stmt> <block> <end stmt>

<if group> ::= <if stmt> [[<then stmt>] <block>]
               {<elif stmt> [[<then stmt>] <block>]}
               [<else stmt> [<block>]]
               <end stmt>

<block> ::= {<simple group> | <while group> | <if group>}

<program> ::= <block> (<end stmt> | <eof>)

<eof> ::= <end of file>
```

3.7.2.2. SEMANTICS

1. A program is terminated upon reaching the first unmatched <end stmt>, or an <eof>. In the latter case, an <eof> acts also as an implicit END for any <while group>'s or <if group>'s still open. Operation of a control structure is the same whether it is explicitly closed or closed by program closure.

2. Any of the program structures <if group>, <while group> may be nested to any depth, so long as the ECI interpreter does not run out of main storage.

3. Well formedness of <group> and <block> structures is only minimally checked by the interpreter. Some anomalies may go unchecked, and may generate unexpected results.

3.7.3. EXPRESSION GRAMMAR

3.7.3.1. SYNTAX

```
<quoted string> ::= <quote> <a string of 0 or more chars
                    other than quote> <matching quote>

<literal string> ::= <a string of one or more chars other
                     than blank, parentheses, or opera-
                     tors as defined below>
<token> ::= (<quoted string> | <literal string>)
          {[<blanks>] (<quoted string> |
                       <literal string>)}

<unary op> ::= [<blanks>] ("+" | "-" | "¬") [<blanks>]

<mulop> ::= [<blanks>] ("*" | "|) [<blanks>]

<addop> ::= [<blanks>] ("+" | "-") [<blanks>]

<relop> ::= [<blanks>] ("¬<" | "<" | "<=" | "¬=" | "=" |
                       ">=" | "¬>" | ">") [<blanks>]

<andop> ::= [<blanks>] "&" [<blanks>]

<orop> ::= [<blanks>] "|" [<blanks>]

<value> ::= <unary op> (<token> | "(" <expression> ")")
          {[<blanks>] ("(" <expression> ")" | <token>)}

<multiply> ::= <value> {<mulop> <value>}

<addition> ::= <multiply> {<addop> <multiply>}

<relational> ::= <addition> {<relop> <addition>}

<and> ::= <relational> {<andop> <relational>}

<expression> ::= <and> {<orop> <and>}
```

3.7.3.2. SEMANTICS

1. For quoted strings both single and double quote marks are supported. An opening quote of either type must be matched by a single closing quote of the same type. Imbedded quotes of the same type must be doubled, but imbedded quotes of the opposite type may be used freely. The actual value of a quoted string is the string itself minus the leading and trailing quotes, and with all imbedded double occurrences of the same kind of quotes, reduced to single occurrences. For example:

```
'abc+2'      has as its value     abc+2
'a''bc+2'    has as its value     a'bc+2
'a''bc""d'   has as its value     a'bc""d
"a''bc""d"   has as its value     a''bc"d
```

2. Literal strings are runs of non-blank characters other than any of the special characters that have been defined as operators (parentheses are considered operators for this purpose). The actual value of a literal string is the string itself. Valid examples of literal strings are:

```
abcz
ab$cd
1234
100ab
!!!#$:;alpha>>>
```

3. The concatenation of the actual values of succeeding quoted and literal strings yields a token. Blanks in between succeeding quoted and literal strings are significant and are concatenated as well. However, leading and trailing blanks are not included. Legitimate tokens are:

```
'a='123              whose value is:  a=123
'a'b'c'd'e'          whose value is:  abcde
'ab' 'de'            whose value is:  ab de
enter command?:      whose value is:  enter command?:
"enter command?:  "  whose value is:  enter command?:
                     (followed by 2 significant blanks)
```

4. For the operators supported by the expression evaluator leading and trailing blanks are disregarded.

5. The basic unit for evaluation of expressions is the <token>. It is nothing more than a string. However, most of the operators are arithmetic and boolean. To support such operations every operand that is the target of an arithmetic operation undergoes an implicit coercion from character string to arithmetic value as follows:

   a) If the character string represents a legitimate signed or unsigned integer then the coercion yields the represented integer.

   b) If the string represents an invalid integer such as +10-2 or if the string contains alphanumeric or special characters, or no digits at all the coercion attempts to extract all digits present in the string, concatenate them, and the resulting integer number is then used in computation. If no digits were present then the value zero is used in computation.

   c) For boolean and relational operands successfully coerced according to rule "a" the value "0" stands for "false" and "¬=0" stands for "true". Otherwise computation proceeds with rule "b". However, relational operands failing coercion rule "a" are then compared using their character values.

6. The result of an arithmetic operation is a signed integer, and the result of a boolean operation is either 0 or 1. For substitution into the original ECI statement, all these values are converted back to character strings.

7. Successive tokens and/or expressions are concatenated just as are quoted and literal strings. The first element of a concatenation may be preceeded by a unary operator. Blanks occurring between succeeding tokens and/or expressions are significant and are concatenated as well.

8. As usual, the order of evaluation of the operators shown in the grammar can be altered by means of parentheses. They may be nested to any desired level that the storage resources of the ECI expression evaluator allow. Examples of expressions are:

```
abc2          yields   abc2
123           yields   123
"a = 2"       yields   a = 2
a = 2         yields   0
'a' = 2       yields   0
'a' = 2       yields   0
'a' ¬= '2'    yields   1
(((-100)))    yields   -100
-(-100)       yields   100
-100.e+2      yields   -100.2
"-100.e+2"    yields   -100.e+2
(1+2)(3*4)    yields   312
(a) (b)       yields   a b
2*(3+4)/2     yields   7
```

9. The result of division by 0 is always 0.

3.7.4. VARIABLE SYMBOL GRAMMAR

3.7.4.1. SYNTAX

<symbol table value> ::= "[" <expression> "]"

3.7.4.2. SEMANTICS

The ECI interpreter supports a symbol table with pairs of the form:

<symbol table name: character string value>

To fetch the value of a symbol the symbol name is enclosed in the reserved bracket symbols "[" and "]". Whenever such bracket symbols are encountered, the symbol name string, denoted by the enclosed expression, is computed and the corresponding symbol table value is substituted in place. This substitution mechanism is implemented in the most general sense:

a) Arbitrary nesting of brackets is allowed, with substitution proceeding from the deepest nested outward.

b) Symbol names are arbitrary expressions.

c) Substitution is done in such a way that the syntax of the surrounding context is not altered. Substituted values are never rescanned, either for substitution purposes or for expression syntax recognition. An exception is "expression lists", defined in the next section.

d) The context surrounding the bracket symbols does not affect the evaluation of the enclosed string; thus quote marks inside a bracketed expression string, all enclosed in quotes do not follow the quote doubling convention; but quotes within matched quotes inside a bracketed expression, all enclosed in quotes, do, regardless of whether all are surrounded by outer level quotes.

e) Since the reserved characters "[" and "]" are normally unavailable, the symbols SYSLDELIM and SYSRDELIM (whose values are "[" and "]", respectively) are provided. Rule "c" guarantees that the substituted values of SYSLDELIM and SYSRDELIM are not rescanned.

f) Symbol table value substitution is concurrent with expression evaluation. Each command evaluates its parameters once upon beginning execution. Reexecution of the command causes reevaluation of its parameters.

g) Symbol names not present in the symbol table yield null values.

A few examples are in order. Given the following symbol table values:

```
n = 2
save3 = list
list2 = 123
```

then both:

```
[[save([n] + 1)] 2] and
[[save([n] + 1)][n]]
```

yield the result "123". Other examples are:

```
"[save"3"]"          yields   list
"[save3]"            yields   list
[save3]              yields   list
"list2 = [list[n]]" yields list2 = 123
[n]+[list2]          yields   125
[[n]]+[[save3](2)] yields 123
```

3.7.5. EXPRESSION LIST GRAMMAR

3.7.5.1. SYNTAX

```
<list> ::= [<expression> {<separator> <expression>}] |
     ["(" <expression> {<separator> <expression>} ")"]
```

3.7.5.2. SEMANTICS

Most command parameters are single expressions; however, some commands have parameters, called sublists, which are themselves lists. Sublists are enclosed in parentheses. Sublist elements are extracted according to the same rules used for command parameter breakout, but sublists do have one unique property. It is possible to alter the number of sublist elements by variable symbol substitution, because symbol substitution occurs before sublist breakout and expression evaluation. This is an exception to the rule that symbol substitution does not alter context.

Since both command parameters and parameter list elements are separated by commas, the convention is made that matched parentheses enclosing ANY command parameter are not a part of the parameter value unless they are inside of quotes.

A few examples follow:

```
1,2,3          yields    1
                         2
                         3

1,1+1,1+1+1    yields    1
                         2
                         3

(1,2),3        yields    1,2
                         3

given
 s1 = xz   98,99

then
 (az  1,2),[s1],,end
                yields    az  1,2
                          xz   98
                          99
                          <null>
                          end

but if
 s1 = "xz  98,99"

then
 (az  1,2),[s1],,end
                yields    az  1,2
                          xz   98,99
                          <null>
                          end
```

3.8. APPENDIX B -- COMMAND SUMMARY

ALLOCATE - allocates an existing data set to an IBM OS file.

(O) return code:
0 --> successful allocation
8 --> unable to locate local data set
16 --> any other error
(I) ddname: O/S 360 file to be bound
(I) dsname: local data set to be bound
(I) share flag: share/non-share semaphore
(I) check: optional CHECK/NOCHECK PO-member test

CREATE - creates an empty local data set.
(O) return code:
0 --> successful creation
4 --> local data set already exists
12 --> insuficient resources
16 --> any other error
(O) dsname: return local data set name
(I) dsname pattern: pattern to generate local dsname
(I) global file type: NSW global file type
(I) filesize: initial file size in kilobits
(I) growth: file growth increment in kilobits
(I) directory size: number of directory blocks

DELETE - deletes a local data set.

(O) return code:
0 --> successful deletion
8 --> unable to locate local data set
12 --> insufficient resources
16 --> any other error
(I) dsname: local data set to be deleted

ELSE - opens an unconditional execute group.

ELIF - opens a conditional execute group.

(I) expression: arbitrary expression whose boolean value determines execution of group

END - closes control structures and programs.

```
EXECUTE   - executes a TSO command.

        (O) return code:
              0 --> program terminated normally
             12 --> program terminated abnormally
        (I) command:  command string to be executed


FREE      - frees an allocated IBM OS file name

        (O) return code:
              0 --> successful free
              8 --> unable to locate ddname
             12 --> insufficient resources
             16 --> any other error
        (I) ddname:  IBM OS file name to free


FREED     - frees an allocated local data set

        (O) return code:
              0 --> successful free
              8 --> unable to locate dsname
             12 --> insufficient resources
             16 --> any other error
        (I) dsname:  local data set name to free


GET       - fetches an NSW file into a local data set.

        (O) return code:
              0 --> successful execution
              4 --> unable to obtain file
        (O) nsw file name:  returns resolved NSW file name
        (O) dsname:  returns local data set name
        (O) altered filespec:  returns filespec if altered
              in any HELP calls, else null .
        (O) qnswcopy:  returns 0 if tool copy made, or
              1 if NSW copy allocated directly
        (O) member name:  returns name of member of library,
              or null
        (I) filespec:  NSW "filespec"
        (I) gft or attrno:  global file type in either
              attribute-code or literal form
        (I) qset:  NSW file semaphore set switch
```

IF - opens a select control structure.

(I) expression: arbitrary expression whose boolean value determines execution of following group

INCLUDE - executes an ECI program from a local data set.

(O) return code:
0 --> successful execution of program
8 --> program aborted during execution
16 --> program could not be located
24 --> any other error
(I) filename: IBM OS ddname bound to a local data set, else null to invoke a default system provided library
(I) member name: PO member to be retrieved from library, else null

INDEX - yields position of argument within a search list.

(O) index: variable to return position 0..n of argument in search list, else -1
(I) argument: character string to be matched
(I) list: search list

INTERPRET - switches the ECI interpreter to interpret mode where ECI commands are executed interactively

MENU - performs menu driven input prompting.

(O) index: returns index of matched input in menu list, or -1 if attention.
(O) input: returns user input
(I) prompt: message to prompt user for input
(I) menu: menu list of acceptable inputs
(I) help: HELP list

NOTE - displays messages on the user's terminal.

(I) message: message to be output on user's terminal

```
PUT      - stores a local data set into the NSW file space.

       (O) return code:
             0 --> successful delivery
             4 --> work's manager won't accept data set
             8 --> local data set can't be located
            12 --> insufficient resources
            16 --> any other error
       (O) nsw file name:  returns NSW file name
       (O) new member name:  returns member name
       (O) new member version:  returns member version no.
       (I) dsname:  local data set name to be delivered
       (I) entry name:  NSW "entry name"
       (I) gft or attrno:  global file type in either
             attribute-code or literal form
       (I) qreplace:  boolean to force NSW file name
             replacement (vs. further disambiguation)

QUERY    - performs prompted input of variable information.

       (O) return code:
             0 --> successful input
            -1 --> user attentioned out
       (O) input:  variable to return user input
       (I) prompt:  message to prompt user for input
       (I) help:  HELP list

SET      - computes  arithmetic and string expressions.

       (O) symbol:  variable to return value of expression
       (I) expression:  arbitrary expression to evaluate

SETQ     - determines whether argument is quoted or not.

       (O) symbol:  returns 1 if argument is quoted, else 0
       (I) character string:  argument character string

SETX     - corrects quoted PO data set names
       (O) symbol:  returns correted dsname
       (I) dsname:  PO data set name to correct
```

STOP - halts interpretation of an ECI program.

(I) abort expression: 1-->abnormal, 0-->normal

SUBSCRIPT - fetches positional element of argument list.

(O) symbol: returns positional element
(I) index: positional index 0..n to apply to list
(I) argument list: list to search

THEN - dummy verb, for commenting IF control structures.

WHILE - opens an iteration control structure.

(I) expression: arbitrary expression whose boolean value determines execution of while group

* - introduces a program comment.

PART IV

BJP/360 -- The NSW MVT Batch Job Processor

This section is separately available
as UCLA document UCNSW-207

4. PART IV: BJP/360

4.1. BJP FUNCTIONAL SPECIFICATIONS

4.1.1. INTRODUCTION

This section of this document has been assembled from working papers prepared by Charles Muntz of Massachusetts Computer Associates, and from discussions held at NSW contractors' meetings. Much of it is derived from references 6 and 7, which we believe to be obsolete.

The Batch Job Package (BJP) on an NSW Batch Tool Bearing Host (BTBH) cooperates with Works Manager Operators (WMO's) to control the execution of NSW batch jobs. Once an NSW user has submitted a job, it is the responsibility of a WMO and a BJP to execute the job and to produce any required status reports. The WMO/BJP combination serves a role analogous to that of the NSW interactive Foreman (reference 1), with a well defined division of the usual Foreman responsibilities.

Specifically, the WMO:

* Serves (through the Interactive Batch Specifier, or IBS -- reference 2) as the user interface to the batch tool.

* Keeps the Local Name Dictionary (LND) for the job.

* Supervises (through the File Package -- reference 3) required file prestaging and delivery operations.

* Optionally, fills information into a skeleton command file and transmits the result to the BTBH for submission.

* Records accounting and statistical information in permanent NSW files.

BJP includes all functions (exclusive of File Package functions -- reference 3) required at a BTBH to accomplish batch job execution. Specifically, it:

* Allocates, manages, and frees the tool workspace.

* Performs the local BTBH Operating System "submit" operation, if appropriate.

* Monitors the submitted job's progress, if requested.

* Optionally forces early termination of the running job.

* Optionally senses job termination and reports it to WMO.

* Reports job time and charges to WMO.

4.1.2. THE BJP-WMO RELATIONSHIP

An NSW may contain any number of WMO's, but not more than one per BTBH. Each WMO consists of a WMO database and any number of WMO processes. Likewise, a BJP may consist of any number of BJP processes, all sharing a single (possibly only virtual) local database. Not more than one such database may occur on a single BTBH within an NSW.

An executing NSW batch job is associated with just one WMO database and just one BJP database. It is therefore associated with a specific WMO host and a specific BJP host. Neither the WMO nor the BJP is concerned with how many processes of the other type are present on the other host associated with a job. All requests in either direction are bound to a specific job, and thus to a specific generic process name on a specific host. Thus all such requests are addressed by host-specific generic process names. Replies to such requests are, of course, specifically addressed.

In other words, when a BJP needs to converse about a specific job, its conversational partner will be any WMO at a specific BTBH. Likewise, when a WMO needs to converse about a specific job, its conversational partner will be any BJP at a specific NSW host.

The conversations that can occur between a job's WMO and BJP are limited to:

* ALLOCATEJOB -- The WMO requests the BJP to allocate a workspace for a new job.

* STARTJOB -- The WMO requests that a fully staged job be submitted to the local BTBH operating system for execution.

* QUERY -- the WMO requests the status of a job.

* JOBHALTED -- the BJP reports to the WMO that a job has been completed.

* ENDJOB -- the WMO requests the BJP to free the job's workspace.

4.1.3. BJP DESIGN GOALS

In designing the BJP external specifications, these goals were considered:

* There will be many implementatons of the BJP, and they will probably all need to work quite differently.

* The WMO's transactions with a BJP to execute a specific tool must be tailored both to the tool and to the BJP that manages the BTBH where the tool is mounted. Thus the WMO will be driven by tables stored in the descriptor for the tool implementation instance, within the Works Manager's data base. The BJP implementor should have a good understanding of the capabilities and limitations of the WMO's table interpretor (reference 4).

* The minimal BJP must be very minimal indeed, as some useful NSW BTBH's will not have the resources to execute or the commitment to implement a massive BJP.

* Specifically, it should be possible for a BJP working with an operating system that keeps sufficient tables of its own to maintain no local database at all.

* Therefore, it must be possible for a BJP to know a job only by its local name.

* On the other hand, a BJP may be as intelligent as it wishes, subject only to the constraints of this specification.

* The WMO will in all cases be the dominant process. It will make the decisions, and the BJP will follow orders.

* The BJP and Foreman specifications should be compatible, and should allow the possibility of future specification of a continuum of Foreman-like processes bridging the gap between the two present specifications.

4.1.4. JOB NAMING SCHEME

Each WMO maintains a queue of jobs in progress. When an NSW user submits a job, the WM to which the user is assigned contacts a WMO with the job request. WMO assigns the job to an unused location in the queue, an assignment which remains intact throughout the stages of job execution.

When a WMO is "cold-started" all its queue entries are marked free. Such cold starts are explicated by maintaining a WMO cycle number, initially 1, which is incremented (except that the successor to 16383 is 1) each time a cold start occurs. Thus the NSW job name is a triple: WMO host number, that WMO's cycle number, and position within that cycle's queue. When WMO contacts BJP regarding an NSW job, its NSW name is included in the message. BJP may, but is not required to, detect changes in the cycle numbers of WMO hosts and initiate local cleanup operations. WMO is incapable of such cleanup, so if the BJP implementation is such that garbage can accumulate from WMO cold starts, this feature may need to be included in the BJP design.

Typically, a BTBH operating system will have its own scheme for assigning a name to a job in its queues, and a BJP will usually have to use that local name when conversing with its local operating system. To support this notion, the BJP may define a "local job name", an arbitrary character string not meaningful to WMO, and may have WMO file that string in its database along with the NSW job name. Such a string may be associated with the job at any point during its life, but once it is assigned, it may not be changed. After such an assignment, every time WMO contacts BJP regarding the job, both the NSW job name and the assigned string are included in the message. When BJP contacts WMO, it can supply either or both names. Thus WMO takes on the responsibility for maintaining the relationship between the names, reducing BJP's need for a local database.

When both names are specified in a BJP-to-WMO message, WMO is committed to make consistency checks. When inconsistent local names are detected by WMO, it will initiate and fully monitor job termination and cleanup activities. In practice, systems crash, and the databases of the BTBH and WMO may not always be in perfect synchronism. It is the responsibility of the BJP designer to guarantee that local-name strings cannot be reused "too soon". Ideally, such names should "never" be reused if there is a possibility that the BJP may contact the WMO and specify the local name only. If the BJP can always provide the full NSW job name, no problem arises so long as the BJP remains self-consistent. More study is needed in this area, but in the meantime, the BJP designer should satisfy himself that his local job naming scheme will not cause synchronization problems.

4.1.5. BJP IMPLEMENTATION OPTIONS

Besides the usual options of program design and implementation that are not externally visible, the BJP external specification allows for much variation in actual BJP behavior. For instance:

* The BJP may be one or many NSW processes.

* The BJP may call its jobs by their NSW names or by their local names. It may assign a local name if it needs to or wishes to, and if it does, it may be assigned at any point during the job's life. It is not absolutely necessary that BJP remember the NSW name of a job.

* The BJP may receive notification from its local operating system when a job is complete, and it may transmit this notification to WMO. Alternatively, WMO can learn of the completion of a job by polling BJP, so it is not absolutely necessary that BJP remember what WMO host submitted a completed job. (But the implementation of the optional notification mechanism does not mean that the required polling mechanism can be left unsupported.)

* It is not necessary that the BJP remember the association between a job and its workspace name.

* The BJP may arrange for job submission in any way that it wishes. The STARTJOB transaction defined in this specification is intended to be only one such mechanism, and the specification can be expanded to accomodate others where a need exists. For instance, on some systems, the File Package may write certain classes of "standard system input" files directly into operating system job queue space. For such a scheme to be useable, the BJP must have some way of knowing the local name for the job, in order to be able to query the Operating System for job status.

* There are at least two mechanisms that can be used to bind the names of pre-staged input files to the executing job. These use WMO/File Package (reference 3) interactions, but the scenarios are set up in the tool descriptor, which must be designed by the BJP designer.

    1) The most general scheme uses "File Package feedback." When the WMO invokes the FP-EXP (reference 3) procedure to request file staging to the BTBH, the File Package returns the generated local file name. This name can be inserted into the command stream by the WMO, and can thus be made available to the tool. The command stream is always the last file to be sent to the BTBH, to allow this scheme to work.

At this writing, File Package feedback cannot be used for the command stream itself. That file is staged through the FP-TRANSP procedure (reference 3), which does not provide such feedback. However, in an implementation that does not use STARTJOB, this might be less of a restriction.

2) A somewhat simpler scheme uses "computable names." In this scheme, each file that the tool may use is assigned a unique and fixed simple name. The BJP ensures that the "workspace" string returned from ALLOCATEJOB is sufficently unique to guarantee that all names formed by combining that string with the set of simple file names are unique. The WMO sends the File Package these fully specified names, and no File-Package-generated names are involved. This is currently the preferred way of handling the command file.

* Likewise, there are at least two mechanisms that can be used to bind the names of deliverable files produced by the tool and delivered by WMO. These, too, use WMO/File Package (reference 3) scenarios which must be designed by the BJP designer.

1) The method currently preferred is identical to the "computable names" scheme mentioned above, except that instead of using FP-EXP to stage data into the computed file names, the WMO uses (indirectly, through WM-DELIVER) FP-IMP (reference 3) to deliver data from those names. The files themselves are created by the executing job.

2) An alternate method, which has not been implemented at this writing, uses an as yet unspecified FP-RESERVE procedure to reserve file space under a given or generated name before tool execution. The files could be named either through File Package feedback or as with computed names.

4.1.6. COMMONLY USED DATA ELEMENTS

This section defines the data lists that are used in the procedure calls defined in the next section.

4.1.6.1. TOOL ID LIST

Every invocaton or reply message between WMO and BJP includes tool id list:

```
LIST(cycle-no,              index, 0 denotes "unknown".
     tool-instance-id,      integer, 0 denotes "unknown".
     local-name)            charstr, length of 0 denotes
                            "unknown".
```

When WMO sends a tool id list it always includes cycle no and tool instance id, and local name if and when WMO learns of it. When BJP sends a tool id list, it must include either cycle no and tool instance id, or local name, or both. If it includes both, and if a local name has been associated with the job before, WMO will perform consistency checks.

Note that this list does not include the "WMO host number" component of the NSW job name. This is because this list is always associated with a generic message, and the MSG primitives "receivegenericmessage" and "sendgenericmessage" provide or require the host number as a separate argument. To include it here would be redundant, and would mereley introduce a potential for inconsistency.

4.1.6.2. ACCOUNTING LIST

The accounting list is as specified for the NSW Interactive Foreman (reference 1).

```
LIST(cost            integer, cost in cents.
     LIST(type,      index, resource type:
                       1 --> CPU seconds
                       2 --> connect minutes
                       3 --> I/O operations
                       4 --> primitive calls
                       5 --> main storage usage
                       6 --> file storage usage
                       .     (to be extended
                       .      according to BTBH
                       .      needs)
          amount)    integer, resource utilization:
     .               in resource units or cents.
     .
     . )        (repeated as many times as needed)
```

This list has two uses. It is transmitted from WMO to BJP in ALLOCATEJOB, where it contains estimates of the resource requirements of a job. In this case, it can be as simple as "LIST()".

It is transmitted from BJP to WMO both in the JOBHALTED message and in the reply to QUERY. In all cases where "status code" is "halted", the values in the list should be the final ones, and they should be equal in all such messages. When more than one copy of such final data reaches WMO, it reserves the right to choose arbitrarily which one to record in its statistical and accounting files. BJP must preserve this data and repeat it in any QUERY reply until ENDJOB has been received.

4.1.6.3. TOOL DEPENDENT PARAMETER LIST

The tool dependent parameter list is as specified for the NSW Interactive Foreman (reference 1). This list is copied unchanged and unexamined from the tool descriptor and sent to the BJP for whatever purposes it may wish.

```
n-LIST(parameter)     charstr -- from tool
                                 descriptor.
```

4.1.6.4. STATUS LIST

The status list is BJP's almost-universal reply to WMO's transactions:

```
LIST(status-code,      index:
                          0 --> not found.
                          1 --> workspace
                                allocated.
                          2 --> scheduled
                                (submitted).
                          3 --> running.
                          4 --> halted
                                (ready for
                                 delivery).
                          5 --> deleted.
                        100 --> job names
                                inconsistent.
                        101 --> job refused
                                by local system
                                (non-recoverable).
     qcan-proceed,     boolean:
                         true -->  proceeding.
                         false-->  has been
                                       cancelled.
     human-oriented-status)
                       charstr, length = 0
                                for null.
```

4.1.6.5. WORKSPACE DESCRIPTOR

The workspace descriptor is as specified for the NSW Interactive Foreman:

```
  LIST(name,            charstr.
       access-info)     charstr, length=0
                                 for null.
```

4.1.7. THE PROCEDURE CALLS

All BJP transactions are with a WMO. All follow the rules of the NSW Transaction Protocol (NTP -- reference 5). All transactions are invoked by host-specific generic process addresses, and are completed, when applicable, by specifically addressed messages.

Following our usual practice, we represent a transaction in the form:

```
G/process-procedure (arguments) -> (results)
```

where "G" indicates that all these transactions are generically addressed, "process" is a shorthand name for the process that receives and executes the transaction, and "procedure" is the name of the specific procedure that that process is to perform. The "result" may be empty if only a positive or negative acknowledgement is defined. In cases where no response of any kind is expected, the arrow and "(result)" are omitted.

4.1.7.1. ALLOCATEJOB

```
G/BJP-ALLOCATEJOB (tool-id-list,
                   accounting-list,
                   tool-dependent-parameter-list)
                -> (tool-id-list,
                   status-list,
                   workspace-descriptor)
```

ALLOCATEJOB is the first communication that a BJP receives about an NSW job. Its successful completion binds a local system workspace to an NSW job name. Depending on local system characteristics, BJP may want to purge the workspace of any existing files. BJP may refuse to allocate a workspace if it believes that the NSW job name is already active. The "accouting list" is used for resource estimates.

4.1.7.2. QUERY

```
G/BJP-QUERY (tool-id-list,
             qproceed)
          -> (tool-id-list,
             status-list,
             accounting-list)
```

"Qproceed" is like "qcan proceed" in the STATUS LIST, with values:

true --> job can proceed.
false --> cancel job.

Cancelling a job merely schedules it for priority exit from the system with a minimum of further resource utilization. The job remains known to the BJP until deleted by a successful ENDJOB. WMO will issue QUERY regardless of whether JOBHALTED is implemented.

4.1.7.3. ENDJOB

```
G/BJP-ENDJOB (tool-id-list,
              workspace-descriptor)
          -> (tool-id-list,
              status-list,
              accounting-list)
```

This is the last call that BJP receives relative to an NSW job. Its successful completion frees the NSW job name, the workspace name, and where applicable, the local job name, for possible reuse, and allows the BJP to discard the accounting information for the job. Depending on local conditions, BJP may also wish to purge the workspace of any remaining files.

BJP may refuse to perform ENDJOB for a job whose status is other than "halted". WMO is obliged to cancel a job in any other state through QUERY, and to wait for it to reach "halted" state before issuing this call.

BJP is not allowed to report normal completion of ENDJOB when the requested job cannot be found.

4.1.7.4. JOBHALTED

```
G/WMO-JOBHALTED (tool-id-list,
                 status-list,
                 acounting-list)
```

This is the only function in WMO invoked by BJP. It requests WMO to initiate job delivery operations. It is an optional feature: if it is not implemented, WMO will eventually discover that the job is completed through the reply to one of its periodic BJP-QUERY requests.

Since there is no acknowledgement to this call, BJP may not assume that WMO has received it until ENDJOB arrives as a confirmation. Accordingly, BJP may, at its discretion, resend this message any number of times. WMO will tolerate such redundant messages until it has received a successful reply to ENDTOOL.

4.1.7.5. STARTJOB

```
G/BJP-STARTJOB (tool-id-list,
                workspace-descriptor,
                filename)
            -> (tool-id-list,
                status-list,
                accounting-list)
```

The "filename" parameter is a character string which fully specifies whatever information the BJP requires to submit the job. Usually, this will be a simple name which, together with the workspace descriptor, identifies and provides access to a command-language file to be submitted to the local operating system.

If the reply to STARTJOB indicates a "status code" of "job refused", then the job proceeds immediately to state "halted". Under present specifications, the STARTJOB may not be retried. WMO will skip delivery and proceed to its job purge scenario.

STARTJOB need not necessarily be a part of a BJP implementation, if other mechanisms can be designed to provide the same facility.

4.2. BJP/360 IMPLEMENTATION

4.2.1. UCLA DEPENDENCIES

BJP/360 is an implementation of the NSW BJP for IBM real-memory systems. Specifically, BJP/360 was developed to operate on the UCLA IBM System/360 Model 91KK under the MVT Operating System with the Time-Sharing Option, TSO (we commonly refer to this combination as OS/MVT).

Unlike other UCLA implementations of NSW components, BJP/360 is not directly exportable to other installations running OS/MVT. The system facilities needed to implement a BJP are not present in OS/MVT except through extensive installation modifications. OS/MVT has been in use for over a decade at this writing, and it exists in many installations. Many of these have added the kind of system facilities needed by BJP; however, there has been little standardization of these additions. Therefore, BJP/360's use of the UCLA system embellishments may or may not map simply into the equivalent embellishments at another installation.

Among the unique system facilities used by BJP are:

4.2.1.1. SYSOUT ROUTING SVC

The UCLA Sysout Routing SVC, or SRS (reference 8) is used to submit data sets containing Job Control Language, or JCL (reference 9). SRS provides services not found in many job-submission facilities, such as:

1) Negotiation of a unique jobname directly, independent of the jobname that will be found in the JCL.

2) Specification of the name of a data set to be created and filled with the standard system output (SYSOUT) that will be produced by the file.

3) Specification of a disk volume where that data set is to be created.

4) Specification of a "message queue" name through which the system is to distribute notification of the fact that the SYSOUT data set has been produced and is ready for viewing.

4.2.1.2. THE DISK SYSOUT WRITER

The UCLA disk SYSOUT writer DSKWTR is used to linearize the various (possibly parallel) output streams produced by the executing job and to place the result in the SYSOUT data set according to the specifications relayed it from SRS. This

mechanism offers these unique facilities:

1) Creation of a data set using a recommended name and a recommended disk volume.

2) Sending notification via a specified "message queue" when the data set is complete.

3) Reporting in the notification message the job name, the cost of the job, and the job's usage of various system resources that are components of billing and statistics.

4.2.1.3. THE GENERAL MESSAGE FACILITY

The UCLA General Message Facility, or GMF (references 11, 12) is used to receive and queue the notifications from DSKWTR. Unique features of GMF include:

1) An interface to using programs that allows them to read and write "messages" to named "message queues".

2) A facility through which a using program can "listen" for notification that messages have been added to specific message queues.

3) Capability to store small numbers (up to 255) of messages in each message queue indefinitely, and to delete messages in any order.

4) A structure that allows multiple using processes to operate on the same message queues concurrently and without any concern for inter-user synchronization.

4.2.1.4. THE MAGIC DATASET AND TABLE

The UCLA "Magic Dataset", or TMD, and "Magic Table", or TMT (reference 13) are used to communicate among the above-mentioned facilities, and also to support job status queries from BJP/360. Unique features of TMT/TMD include:

1) Extensibility of the OS/MVT job queue -- TMD can be considered a direct extension to the "job table" kept in the job queue. It thus provides additional space for storing job information of kinds that the OS/MVT facilities do not support.

2) Fast access to job information -- TMT is an in-storage summary of the job tables in the job queue and TMD.

3) Up-to-date information on a job's status -- any program with the proper authorizations can inquire of TMT as to the (almost) instantaneous state of a queued or executing job.

4.2.2. COMMUNICATIONS SUMMARY

BJP/360 functions as an NSW core-system process with generic name "BJP". It communicates with other NSW processes using the NSW Network Transaction Protocol, or NTP (reference 5) On an IBM system, NTP is implemented on three levels:

* The procedure-call level is implemented by the PL/PCP subroutine package (reference 15).

* The MSG message level is implemented by the PL/MSG subroutine package (reference 16).

* The NSWB8 data encodement level is handled by the PL/B8 subroutine package (reference 10).

4.2.3. OPTION CHOICES

In designing a BJP implementation, certain choices in behavior are allowed. In the case of BJP/360, these options have been selected:

4.2.3.1. The BJP is a single task, and executes as a single swapped time-sharing job under TSO. It is written in PL/I (IBM Optimizing Compiler), with a few Assembler-language subroutines for system interfaces.

4.2.3.2. The BJP is automatically started by MSG Central whenever MSG is initialized. Subsequently, should the BJP crash in any way, it will be restarted through the normal process-spawning capabilities of generic-message processing.

4.2.3.3. BJP/360 materializes to MSG as a single NSW process, which accepts only generic messages, and will accept any number of them. MSG is aware, through the local attributes attached to generic name "BJP", that only one such process is ever to be started at a time. Thus, if a generic message for a BJP arrives at a time when a BJP process exists but is not enabled to receive it, the message will be queued for that process, rather than trigger spawning of a new process.

4.2.3.4. BJP/360 assigns a unique local jobname at the time of processing the STARTJOB transaction, so a successful reply to that transaction will always include a local name.

4.2.3.5. BJP/360 does not keep a formal job table; however, the facilities of SRS, DSKWTR, GMF, TMD, and OS/MVT allow it to recover the WMO host number, NSW job number, and workspace name when it is notified of a job's completion. A job can be located either by its NSW name or by its local name during much of its processing, as in this summary:

	Can locate by local name?	Can locate by NSW job number?
after ALLOCATEJOB		
(same BJP)	No	Yes
(new BJP)	No	No
after STARTJOB		
(same BJP)	Yes	Yes
(new BJP)	Yes	No
after JOBHALTED		
(same BJP)	Yes	Yes
(new BJP)	Yes	Yes

4.2.3.6. BJP/360 notifies the WMO of job completion through the asynchronous JOBHALTED message, always specifying both the local and NSW job names. Of course, it also supports polling.

4.2.3.7. BJP/360 uses STARTJOB as the mechanism for submitting a job to OS/MVT. The job is contained in a local data set built by and filled in by the WMO (using File Package FP/360, of course -- reference 3). BJP/360 never examines or modifies the contents of this file. This means that all knowledge of OS/MVT JCL is concentrated in the Tool Descriptor and the WMO.

4.2.3.8. In the general case, BJP/360 does not participate directly in the mechanisms for binding the names of input and output files to the executing job. There is one notable exception; the SYSOUT file uses the "computable names" scheme. Its name is always "<workspace>.SYSOUT".

4.2.4. LOCAL JOB NAMES

When a job is submitted by the STARTJOB transaction, it is defined by a Job Control Langauge (JCL -- reference 9) data set. Normally, the JOB card in this data set would contain the local job name. Because OS/MVT requires that jobnames be unique, and because UCLA requires that the first six characters of the job name be the job's charge number, the SRS facility includes the capability to specify a job name external to the data set being submitted. BJP/360 uses this capability to negotiate a unique jobname with OS/MVT. It starts with a name of the form

cccccxx

where "cccccc" is a charge number gotten from a BJP/360 initialization parameter. "xx" represents two "wild characters" which are filled in with the two low-order base-32 digits of a pseudo-random number. BJP/360 attempts to submit the job with this name. If job submission fails with an indication that the job name is not unique, then "xx" is incremented by 1, taken modulo 32**2, and used to make a new job name. The job is resubmitted with this name. This process continues until either the job is submitted, the job fails for a reason other than non-uniqueness, or until all 32**2 combinations have been tried in vain. If a job name is successfully negotiated, it is returned to the WMO, which must use it in any status queries that might cause TMT interrogation.

4.2.5. WORKSPACE MANAGEMENT

Because BJP/360 supports the "computable names" scheme for file prestaging and delivery, it is required to quarantee that the workspace name returned by ALLOCATEJOB be unique, and that it designate an empty file directory. To accomplish this, it defines a workspace name to be of the form

aaaaaa.iii.scccccc.sjjjjjj

where:

"aaaaaa" is the UCLA charge number under which BJP/360 is to allocate workspaces. It is gotten as an initialization parameter.

"iii" is the UCLA user "initials" under which BJP/360 is to allocate workspaces. It is gotten as an initialization parameter.

"scccccc" is the GMF queue name, as defined in the section entitled "OPTION CHOICES".

"sjjjjjj" is an encodement of the pair ("NSW system", "NSW job number"). "s" is the same character used in the GMF queue name. "jjjjjj" is the seven-digit base-32 representation of the NSW job number (also called the "tool-instance identifier").

The resulting directory is guaranteed to be unique so long as the 4-tuple (NSW system, NSW host, WMO cycle number, NSW job number) is unique. Thus the burden of uniqueness is shifted to the WMO's.

The requirement that the workspace name designate an empty file directory is met by issuing the TSO command

DELETE <workspace-name>.*

against the directory. In fact, this command will not delete data sets with "." characters in the part of the name corresponding to the "wild" character "*"; however, the WMO normally does not generate such names, so this is not likely to be a problem. A better mechanism should be considered for future implementatons.

The workspace is created by ALLOCATEJOB and destroyed by ENDJOB. In OS/MVT the corresponding file directory will be created when the first file is entered into it, usually in the process of prestaging tool files. Ideally, the directory should be emptied and deleted by ENDJOB; however, this is not being done at this writing. Eventually, the directory should be emptied by the same mechanism used to guarantee that it is initially empty. It should be

destroyed by the DELINDX entry of the PLIDAIR package (reference 14), which did not exist when BJP/360 was written.

4.2.6. MANAGING CYCLE NUMBERS

When BJP/360 notices a change of cycle number for any given WMO, it should instigate a cleanup operation. The present implementation only writes a log message when this is detected. A complete implementation would perform at least these cleanups:

* Scan OS/MVT for any jobs belonging to obsolete cycle numbers of the WMO host in question. If any are found, cancel them in such a way that, should GMF notification for them arrive at BJP/360, it will be ignored.

* Scan GMF for all queues for the WMO host, but for a cycle number different than the one noted. Delete all entries of such queues, and then delete the queues themselves.

* Scan the filespace defined by the "charge number" and "initials" being used to create workspace names, and for each subspace with a directory name that indicates an obsolete cycle number of the WMO host in question, delete all files in the directory, and delete the directory itself.

* Since all these cleanups cannot happen simultaneously in an actual running system, leave things in such a state that the cleanup will continue to be repeated until all obsolete objects are successfully deleted.

4.2.7. JOB TRACKING

BJP/360 does not keep a formal job table, in keeping with the BJP design goals. However, there are three mechanisms in the implementation that approximate a job table:

4.2.7.1. THE TMT ENTRY

From the time that a job is successfully submitted until the time that its SYSOUT file is created by DSKWTR, a job is represented in the UCLA implementation of OS/MVT by a TMT entry. During this time, BJP/360 can interrogate the status of the job through TMT query services. The job must be identified by its local name, not by its NSW job number. The information that can be made available includes:

* The stage of OS/MVT job processing in which the job now resides, such as input queue, execution, or output queue.

* The name of the work queue containing the job, when applicable, and its position in the queue.

* The amount of system resources used so far by an executing job, including its cost so far.

4.2.7.2. THE GMF QUEUE ENTRY

Between the time that DSKWTR notifies BJP/360 that a job's SYSOUT data set is ready and the time that BJP/360 processes an ENDJOB for the job, the job is represented in a GMF message queue. The particular queue used is specific to the WMO database to which the job belongs. Thus the logical queue name is the triple:

(NSW system, WMO host, WMO cycle number)

where "NSW system" is used to keep separate the various NSW's that can be running on the UCLA host simultaneously (User system, Candidate system, Development system, etc.). Since GMF queues are assigned 8-character alphanumeric names, this logical name is mapped onto a real name of the form

scccccc

where:

"s" identifies the NSW system, and is gotten from the BJP/360 initialization parameters. By convention, we use the three EBCDIC "national" characters, "@", "#", and "$" for this purpose, since they never occur as the leading characters of queue names created by other GMF users. (There is no fencing of GMF queue name space for NSW.)

"ccccccc" is a seven-digit base-32 number (the base-32 digits are defined to be A-Z and 0-9, excluding "I", "O", "0", and "1"). It thus carries the same significance as a 35-bit binary number. The value of the number is:

(nsw host number) * 65536 + (WMO cycle number)

The contents of a "message" in a GMF queue include:

* The charge number associated with the job.

* The TSO initials, if any, associated with the job.

* The name of the SYSOUT data set.

* The volume on which the SYSOUT data set resides (this can also be determined from the system catalog).

* The local job name.

* The cost of the job.

* Values of the job's consumption of various system resources that are components of job billing.

An important side effect of the selected workspace naming scheme allows much information about a job to be recovered from its GMF queue entry. This is significant when the incarnation of BJP/360 reading the GMF queue is not the one that submitted the job. A part of the data in the GMF queue is the name of the SYSOUT data set. This name can be parsed to yield the workspace name, which can be parsed to yield the NSW system, the WMO host number, the WMO cycle number, and the NSW job number. Another part of the data gives the local job name directly.

The set of currently defined GMF queues is summarized by an main-storage table containing for each queue name:

* The queue name.

* The binary host number and cycle number.

* A pointer to the last-read entry in the queue.

* Various values used by GMF to manage the queue.

4.2.7.3. THE SHORT-TERM MEMORY

Because the two tables mentioned do not span the entire lifetime of a job, and because there are instants when a job is in transition between those tables, BJP/360 keeps an internal "short-term memory" table. This table contains all those jobs that have been mentioned in transactions during this incarnation of BJP/360, as well as those represented in GMF entries when BJP/360 initialized. It is thus highly likely to contain an entry for an old job mentioned in a new transaction. For each known job, this table contains:

* The job's NSW name -- the host number, cycle number, and job number.

* The job's local name.

* The time the job's status was last updated.

* The status code at last update.

* The status message at last update.

* The last value of "qproceed".

* The values for the *accounting* list.

* A pointer to the job's entry in the GMF queues.

The purpose of the status information and its timestamp is to support a feature not implemented at this writing. Since there are instants when a job is in transition between TMT and GMF, if a status query fails to locate a job, it can be said to be in the state last tabled here. If, however, after an amount of time specified by a BJP/360 initialization parameter, the job is still not found, it should be assumed to be lost.

4.2.8. LOGIC SUMMARY

The major routines of BJP/360 are summarized below

BJDISP is the mainline of BJP/360. It materializes the BJP process, which remains alive until an MSG terminationsignal is received, or until the process has been idle for an amount of time gotten from a BJP/360 initialization parameter. During the lifetime of the process, BJDISP waits for and processes one event at a time, with no attempt at simultaneity. Events are of three types: NSW transactions, GMF notifications, and timer expirations. For each event that occurs, an appropriate processing subroutine is selected and called.

BJENDC exists for the purpose of isolating the relationships between the GMF queue names and workspace names and the things out of which they are made.

BJENLST isolates the encodement of common list structures to one routine. There are separate entries to encode the tool id list, the workspace descriptor, the accouting list, and the status list.

BJENRPL is a short piece of very common code to encode output values and replies to a transaction.

BJFNDJ locates or builds an entry in the short-term memory table for a given job.

BJFNDQ locates or builds an entry in the main-storage GMF-queue summary table, corresponding to a given queue name. If a change of cycle number is detected, BJNEWQ is called.

BJGETP reads the BJP/360 initialization parameters.

BJGMFCL closes out GMF in response to an error or a BJP shutdown.

BJGMFOP attempts to open or re-open the GMF connection. When it is successful, it locates each BJP queue and establishes a notification link to that queue.

BJINIT initializes common data areas.

BJNEWQ schedules the cleanup of the GMF queue space and the data set name space when a change of cycle number is detected. In the current implementation, it is only a stub that logs a message about the need for a cleanup.

BJPALJ processes the ALLOCATEJOB transaction.

BJPEND processes the ENDJOB transaction.

BJPLID parses the tool id list that is the first element of every incoming transaction.

BJPQRY processes the QUERY transaction.

BJPSTRT processes the STARTJOB transaction.

BJREJEC builds an error message and transmits it in response to a remote transaction directed toward the BJP.

BJXNOT reads GMF messages, updates the internal tables accordingly, and sends the JOBHALTED transaction to the WMO.

BJXPCP processes transaction events as signalled through the PL/PCP package (reference 15). For "CALL" type events, it parses enough of the transaction to identify the job being spoken of and calls the appropriate procedure processor.

BJXTIM maintains an outstanding timer interval of a duration specified by an initialization parameter. On expiration of the interval, it does three things:

* If GMF is not open, it simulates a GMF notification, which will result in an attempt to re-open GMF.

* If enough idle time has gone by, and if there are no jobs in the short-term memory table, it initiates BJP shutdown.

* If shutdown is not to be initiated, it sets up another timer interval.

4.3. APPENDIX: BJP/360 INITIALIZATION PARAMETERS

BJP/360 decodes a set of initialization parameters from a configuration data set which may optionally be supplied under file name (DDNAME) PARMS. This data set is in the form of a PL/I GET DATA input stream. The following data may be specified, where each name should be qualified by the name "P.":

Name:	Type:	Default:	Meaning:
TMT_TIMEOUT_MINS	FIXED	5	Minutes to allow a job to be "not found" before calling it "lost". The mechanism to use this datum is unimplemented.
MAX_MINS_IDLE	FIXED	30	The amount of time to allow BJP/360 to be idle before shutting it down.
NAP_INTERVAL	FIXED	30000	The timer interval, in .01 seconds.
MSG_TIMEOUT	FIXED	6000	The MSG message timeout (not the PL/PCP transaction timeout), in .01 seconds.
GMT_ADJUSTMENT	FIXED	8.0	Number of hours EARLIER than Greenwich to assume the local clock to be running. May be signed and may carry a fraction of ".0" or ".5".
SIZE_QUEUE_LIST	FIXED	10	Number of entries in the internal GMF queue summary table. Places a bound on the number of WMO's that can be handled.

(continued)

Name:	Type:	Default:	Meaning:
SIZE_JOB_LIST	FIXED	100	Number of entries in the internal short-term-memory table. Places a bound on the number of jobs that can be handled at one time.
GENERIC_NAME	CHAR	'BJP'	BJP/360's generic name.
CHARGE_NUMBER	CHAR	'NSW001'	UCLA charge number to use for submitting jobs and for making up workspace names.
INITIALS	CHAR	'NWK'	TSO user initials to use for making up workspace names.
SUBMIT_PASSWORD	CHAR	'PROTOCOL'	Password associated with UCLA charge number to use for submitting jobs.
PREFIX	CHAR	'$'	Unique initial character to identify NSW system, for making up GMF queue names and workspace names.
OUTPUT_VOLUME	CHAR	'NSWP01'	Volume on which to request DSKWTR to write SYSOUT data sets.
SUBMIT_DESTINATION	CHAR	'U' ;	"Destination" parameter to give to SRS when submitting jobs. 'U' instructs OS//MVT to deliver the job's output to DSKWTR.

REFERENCES

[1] Schantz and Millstein, "The Foreman: Providing the Program Execution Environment for the National Software Works." BBN document 3442 and MCA document CADD-7701-0111, January 1, 1977. References to this document are as it has been amended by various network messages and verbal agreements, particularly the message from Schantz to Braden dated July 11, 1977.

[2] Bolduc, "IBS meta-language Facilities". MCA unpublished working paper, 1980.

[3] Braden and Ludlam, "FP/360 -- The NSW MVT File Package". UCLA document UCNSW-204, November 20, 1980.

[4] Sluizer, "The Works Manager Procedures: Externally Callable Routines in the Works Manager Maintenance Manual". MCA document CADD-7906-0118, June 1, 1979.

[5] Massachusetts Computer Associates, "The A-Level System Specification for the National Software Works (preliminary)". May 15, 1979.

[6] Muntz, "Batch Job Package -- External Specification." MCA working paper, January, 1978.

[7] Muntz, "Batch Job Package -- Functional Description and Transmission Formats." MCA working paper, March 16, 1978.

[8] Rivas, "using the System Routine SVC to Submit Jobs". UCLA document S-179, August 12, 1975.

[9] IBM Corporation, "IBM System/360 Operating System: Job Control Language Reference". IBM order no. GC28-6704, 1976.

[10] Braden, "PL/B8 -- A PL/I Interface Package for NSWB8". UCLA document UCNSW-403, November 15, 1980.

[11] Worth, "Programming Using the General Message Facility". UCLA document S-180, August 14, 1975.

[12] Braden, "PL/GMF -- PLIX Interface to GMF". UCLA document S-182, August 14, 1975.

[13] Braden, et. al., "An Implementation of MVT". UCLA document TR-1, August, 1969.

[14] DeLa Roca and Ludlam, "PLIDAIR -- Dynamic Allocation from PL/I". UCLA document UCNSW-407, February 11, 1980.

[15] Ludlam, "PL/PCP -- An NSW Procedure Call Protocol Package for PL/I". UCLA document UCNSW-402, November 15, 1980.

[16] Ludlam and Rivas, "PL/MSG -- An MSG Interface for PL/I". UCLA document UCNSW-401, November 15, 1980.

PART V

Configuration Management
for the Development Execution Environment

This section is separately available
as UCLA document UCNSW-208

TABLE OF CONTENTS

5. PART V: CONFIGURATION MANAGEMENT

5.1. ENVIRONMENT OF INTEREST

In the NSW Development System, configuration management of the execution environment is primarily concerned with only two files -- a library of executable programs usually called SCMDLIB, and one of canned procedures usually called SCMDPROC. In the schemes being proposed here, source code is excluded purely as a matter of the limiting of scope. Tool-related data is excluded as being independent of the configurations that we need to control. MSG-central is excluded because, since the UCLA implementation is presently under the control of the local Operating Systems group, its configuration is outside the control of NSW.

This document considers only the Development system; however, almost everything said here applies equally well to all NSW systems. Any configuration management scheme must be designed to be useful in the context of the group of four such systems currently maintained at UCLA. These systems share some source code.

5.1.1. CONTENTS OF FILE "SCMDLIB"

BJP

This module is the Batch Job Processor.

FM

This module is the Foreman mainline.

FMECIMG

This module is the Foreman's Encapsulator Command Interpretor (ECI) subcomponent.

FMINIT

This module reads the Foreman's configuration file and sets internal data accordingly.

FP

This module is the File Package mainline.

FPBCMGR

This module is the File Package's Basic Copy Machine subcomponent.

FPGTTAB

This module is the Global Type Table as referenced by both the File Package and the ECI.

FPINIT

This module reads the File Package's configuration file and sets internal data accordingly.

MSGSTAX (alias MSGCMD, MSGEDIT)

This module is the separately-loadable parts of the PL/MSG and MSGBUG packages

M2

This module is the passive half of the M1/M2 pair of MSG measurement processes.

TCAM

This module is the passive half of the UTEL/TCAM pair of processes for testing the special UCLA TCAM direct-connections.

T2

This module is the passive half of the T1/T2 pair of MSG exercising processes.

T4

This module is the passive half of the T3/T4 pair of direct-connection exercising processes.

5.1.2. CONTENTS OF FILE "SCMDPROC"

BJP

This procedure drives the Batch Job Processor.

FIRST

This procedure is executed when any NSW TSO server process is logged on. It sets up an initial "USE" library.

FLPKG

This procedure drives the File Package as an MSG-initiated process.

FOREMAN

This procedure drives the Foreman as an MSG-initiated process.

LIVEFM

This procedure drives the Foreman as a terminal-initiated process.

LIVEFP

This procedure drives the File Package as a terminal-initiated process

M2

This procedure drives the M2 measurement process.

PBJP

This text segment is the BJP's configuration file.

PFM

This text segment is the Foreman's configuration file.

PFP

This text segment is the File Package's configuration file.

TCAM

This procedure drives the TCAM test process.

TESTFM

This procedure drives the Foreman as a terminal-initiated process under the TSO TEST debugger.

TESTFP

This procedure drives the File Package as a terminal-initiated process under the TSO TEST debugger.

T2

This procedure drives the T2 test process.

T4

This procedure drives the T4 test process.

5.2. PRESENT CAPABILITIES

In the present system, configuration management has to be done completely manually, since no aids to such a process exist. The scheme used can consist only of reccommended procedures, without any enforcement mechanisms; therefore, it will work only to the extent that TBH programmers toe the line.

5.2.1. LOGGING ON

The UCLA Development NSW is stored under and operates from directory AHA183.NWT. In order to operate on Development NSW data with proper automatic data set name prefixing, it is necessary to log onto TSO under this directory. Assuming ARPANET access, such a session will follow this skeleton (upper case is system output, lower case is user input):

```
UCLA OAC 3033  SERVER TELNET     <VERSION STAMP>
 <BROADCAST MESSAGES>
ENTER COMMAND OR 'HELP'
logon nwt/<password>
IKJ56455I NWT LOGON IN PROGRESS AT <TIME AND DATE>
WELCOME TO UCLA-OAC TSO
READY:  <user commands>
          .
          .
          .
READY:  logoff
<RUN STATISTICS>
IKJ56470I NWT LOGGED OFF TSO AT <TIME AND DATE>
```

For "<password>" substitute the current UCLA Development NSW password string, which qualified personnel can get from the UCLA NSW development staff.

5.2.2. UPDATE HISTORY RECORDING

The main vehicle for update history recording is a sequential text file, named CONFIG, in the NSW system being managed. Whenever an update is made to the system, an entry is added to CONFIG to document the change. This entry is edited in over a standard template, including:

* Date and time of system modification.
* The simple name of the library modified (SCMDPROC or SCMDLIB).
* The member name.
* Person identification of modifier.
* A prose explanation of the reasons for and the impact of the change.

Whenever a new entry is added to CONFIG, it is placed at the beginning, effectively pushing the other entries down. Thus the file is maintained in inverse chronological sort. Each new entry is also transmitted to ACC, via network mail, as a notification of change.

CONFIG is updated only by TBH personnel, but it can be examined by ACC. To do so, ACC logs onto UCLA TSO and enters the command

```
DISPLAY CONFIG
```

DISPLAY accepts the same subcommands as the NSW tool named DISPLAY-UC [ref. 1].

5.2.3. PRIMITIVE VERSION NUMBERING

Since there is no facility for keeping multiple versions of a single member in the same library, UCLA has adopted an informal convention for identifying current, pending, and obsolete versions of members. If 'x' is a member name, then the name 'x@' refers to the most recently displaced version. Likewise, 'x@@' or 'x@2' is next oldest, 'X@3' is next, and, in general, 'x@n' is 'nth' oldest. If 'x$' exists, it is a version pending installation (note that "$" modules are not yet represented in CONFIG). If 'x' has more than six characters, then it may be truncated on the right to make room for modifiers. This requires that names not depend on the seventh and eighth characters for uniqueness.

This scheme requires that each update involve renaming the entire set of related members. It has worked in the past only because we have typically kept only one or two old versions.

A typical directory, viewed with the 'PDS' processor (the same as NSW tool LIBMAINT-UC [ref. 2]), might look this way:

```
pds scmdlib
PDS: d
BJP         BJP@         FM          FM@         FMECIMG     FMECIMG@
FP          FP@          FPBCMGR     FPBCMGR@    FPGTTAB     FPGTTAB@
FPINIT      MSGCMD  -A   MSGEDIT -A  MSGSTAX     M2          TCAM
T2          T4
```

5.2.4. PRIMITIVE ENFORCEMENT

At present, no enforcement mechanisms can be provided for SCMDPROC; however, certain features of IBM load modules and the Linkage Editor program can be used to provide minimal policing for SCMDLIB. With each load module stored, there is associated a "last link-edit" date. Each subsequent binary patch associates a control-section id and patch date with the load module. These associated data can be listed with the PDS HISTORY subcommand, thus giving ACC a mechanism for validating the contents of CONFIG. For example:

```
PDS: hi bjp
HISTORY SUMMARY FOR MODULE BJP
LAST LINK-EDITED ON  5/07/79
IMASPZAP UPDATE HISTORY BY CSECT -
BJDFLT          6/26/79     TSO--HCL
BJDFLT          3/01/79     TSO--HCL
BJDFLT          3/01/79     TSO--HCL
BJDFLT          3/01/79     TSO--HCL
BJDFLT          1/22/79     TSO--HCL
GMOPEN#         5/11/78     TSO--HCL
BJOVMSG         5/23/78     TSO--HCL
PDS: hi bjp@
HISTORY SUMMARY FOR MODULE BJP@
LAST LINK-EDITED ON  8/14/78
IMASPZAP UPDATE HISTORY BY CSECT -
BJDFLT          3/01/79     TSO--HCL
BJDFLT          3/01/79     TSO--HCL
BJDFLT          3/01/79     TSO--HCL
BJDFLT          1/22/79     TSO--HCL
GMOPEN#         5/11/78     TSO--HCL
BJOVMSG         5/23/78     TSO--HCL
```

5.2.5. PRIMITIVE DERIVATION TRACING

It is possible to maintain a primitive derivation trace within each load module, and to reference it through the PDS HISTORY subcommand. For each control section of a stored load module, there can exist up to forty characters of "user-supplied update history". This information is specified directly on linkage-editor control cards. It can be replaced, but not added to, by subsequent linkage edits. If it is not replaced, it will be propagated through the linkage editor, and will still be available in any load module of which the original control section is a part.

Each compiled program can be stored with the same sort of data as in CONFIG, less the prose, in this history string. Then the PDS HISTORY command will list all that data for every NSW-produced control section in a load module of SCMDLIB. This provides an mapping back from the executable program to the compiled routines which make it up. The linkage editor control cards to store this data can be supplied automatically by the existing NSW maintenance tools. A sample PDS output (not from SCMDLIB) follows:

```
PDS: hi objscan
HISTORY SUMMARY FOR MODULE OBJSCAN
LAST LINK-EDITED ON  1/31/80
USER-SUPPLIED UPDATE HISTORY BY CSECT -
PLISTART     1/31/80   OBJSCAN   JAN3080 152521 HCL AHA183.NWX
PLIMAIN      1/31/80   OBJSCAN   JAN3080 152521 HCL AHA183.NWX
INFILE       1/31/80   OBJSCAN   JAN3080 152521 HCL AHA183.NWX
OUTFILE      1/31/80   OBJSCAN   JAN3080 152521 HCL AHA183.NWX
SYSPINT      1/31/80   OBJSCAN   JAN3080 152521 HCL AHA183.NWX
OBJSCAN2     1/31/80   OBJSCAN   JAN3080 152521 HCL AHA183.NWX
IELCGMV      1/31/80   OBJSCAN   JAN3080 152521 HCL AHA183.NWX
OBJSCAN1     1/31/80   OBJSCAN   JAN3080 152521 HCL AHA183.NWX
```

This example shows a module produced by a single assembly and a single link; however, most NSW modules would show more variation in the data. In a more complex module, the listing of this history data can become quite massive. It can be halted by entering ctl-C or the TELNET IP sequence. That should return the session, as soon as buffers have emptied, to the "PDS" prompt control level.

The fields shown are:

* Control section or common block name.

* Date of linkage-edit that inserted this history line.

* Name of load module produced by that linkage edit (in NSW, usually the same as the source module).

* Date of submission of compilation job.

* Time of submission of compilation job.

* Id of Person submitting compilation job.

* Directory under which compilation was submitted. In this example, the module was compiled under the Debug NSW (NWX), as will frequently be the case for control sections that find their way into the Development NSW.

5.3. AN INTERIM CONFIGURATION MANAGEMENT PROPOSAL

This section is a proposal for comment only. It is not an offer or commitment to produce a system.

Interim Configuration Management, or ICM, is a scheme to allow NSW ACC to exercise minimal essential configuration management control over the UCLA TBH implementation software. At the same time, it provides the basic hooks for functions that local software development and maintenance personnel need to keep track of software updates. At this point, the scheme is a rough proposal, and it should be expected to change in detail, if not in concept.

ICM proposes that all UCLA NSW libraries be managed with special tools and/or procedures that will make them appear to support multiple versions of member names. These versions will carry absolute version numbers and creation stamps. There will be procedures for manipulating these members and versions, and for aging obsolete versions to offline storage without destroying them.

The ICM procedures are only aids, and are not intended to be enforcing mechanisms. They are not complete enough to form a closed, rigid system, and so will have to be subverted on occasion in order to get the job done.

5.3.1. BASIC CONCEPTS

* All data, text or binary, will be stored in PDS's in order to take advantage of the user-defined attributes that they provide. This is already being done for other reasons.

* Every member will have associated with it a member name and an absolute version number. Multiple versions of the same member name may exist in the same library at the same time.

* Version numbering will be accomplished by storing all data under special "invisible names" in the PDS. The "visible name" of a member will be an alias of just one of these names.

* The "visible" member name will refer to the "current" (latest) version of the member. Under this name, it is available to any program, just as without ICM. Other versions are accessible only through special tools.

* Version numbers are meaningful only in the context of a given PDS. Copying a member from one file to another does not preserve the version number sequence from the input library. Therefore, installation by renaming an entire library should not be encouraged.

* Normally, when a member is "replaced", its previous version is pushed down. This means that the data associated with its previous version remains associated with the "hidden name" of that version. The member name, along with a new "hidden name" become associated with the new version.

* However, it will be possible to actually replace the current version. In this case, both the member name and the "invisible" name are altered to refer to the new data.

* Associated with each version of each member is a "version stamp". This identifies the JOB that created the version. It consists of: 1) the date and time of the creation of the job; 2) the initial member name under which the data was stored; 3) the user identification of the creator of the job; 4) the NSW system identifier of the NSW system under which the job was run (as the Development or Debug system).

* The version stamp will be treated specially for load modules. It will be associated both with the library member and with each control section that occurs within the program. The Linkage Editor will propagate the latter association into any executable program which includes the member. It will thus be possible to "map" such a program and list the version stamp of each of its components. This will be an aid to debugging as well as to configuration management.

* Any library may have portions aged to offline storage. In this case, obsolete versions of some members may not actually be present in the PDS. To keep track of these members, a PDS may contain special "backup" members which contain, instead of data, lists of members that have been demoted to offline storage, and some information on where to look for them. These members can be accessed by special tools.

5.3.2. USAGE IN THE EXECUTION ENVIRONMENT

The outputs of all compilers will be stored with version numbers and stamps. The executables linked into SCMDLIB will carry version numbers and stamps, as well as propagating the stamps of all component routines. There will be provisions for listing the version-related data in a PDS directory, and for mapping a load module with version stamps listed per control section. Since creation of load modules is already done using NSW-specific tools, these procedures are not likely to be subverted through carelessness.

Update of SCMDPROC will be done only through special EXTRACT and FILE tools, so that all members will carry version numbers and stamps. Since this kind of update is normally performed by a naked editor, it will take a certain increase in programmer discipline to avoid subverting the system in the case of this library. Fortunately, however, updates to this library are less common than to SCMDLIB.

Updates will still be reported using the CONFIG file and network mail, as described in section 1.

A backup-and-purge tool will be provided, but it must be manually initiated. No automatic handling of exploding file problems will be provided.

Re-creation of a previous configuration can be done with standard IBM and UCLA utility programs, along with some special programs provided for ICM. Such re-creation should always create new version numbers, even if this means that the same PDS member has two version numbers, and thus two "hidden names".

5.3.3. PROGRAMS REQUIRED

A full implementation of this interim scheme would require writing or modifying several programs or procedures, some trivial, some not so much so.

* EXTRACT MEMBER -- Given a library name, a member name, an optional version number, and an output file name, extract a member's data into a sequential file or an uncontrolled library. This program should be trivial.

* FILE MEMBER -- given the name of a sequential file or an uncontrolled PDS member, a library name, and a member name, store the input data in the library as the current version of the member, providing a version stamp. There should be an option to replace or push down the member. This program should be trivial.

* LINK-EDIT POSTPROCESSOR Post-process a library into which new programs have just been link-edited, to simulate the FILE MEMBER program's action. This might well be the same program, used in a slightly different way.

* BACKUP AND PURGE -- Copy to an offline file those members of a library which exceed a certain age and are not current. Add to the library a special "backup" member which contains a list of those versions copied. Then delete the versions and garbage-collect the file. This program is simple in concept, and if enough existing subprograms can be used unchanged, it will be simple to implement.

* UTILITY MODIFICATIONS -- The existing PDS utility program (NSW tool LIBMAINT-UC) should be modified to accept and report member names in the form of visible names and version numbers, and to group multiple versions together as is done by TENEX. Since this is an existing program of some complexity, this might be a complex task.

APPENDIX -- SAMPLE "CONFIG" FILE

```
        NOTE -- TABSET ON(10 18 26 37 43)
10/12/79 00:00   SCMDLIB (FPBCMGR)  HCL   CORRECT INVALID OVERLAY.
10/05/79 00:00   SCMDLIB (FP)       HCL   CHANGE ERROR CLASS FOR WMO.
09/28/79 00:00   SCMDLIB (FM)       HCL   STOP FALSE LNDSAVE'S.
09/13/79 00:00   SCMDLIB (FPGTTAB)  HCL   RELEASE 4.1.
08/15/79 00:00   SCMDLIB (FM)       HCL   RELEASE 4.1.
08/14/79 00:00   SCMDLIB (FMECIMG)  HCL   RELEASE 4.1.
08/07/79 00:00   SCMDLIB (FPBCMGR)  HCL   RELEASE 4.1.
08/03/79 00:00   SCMDLIB (FP)       HCL   RELEASE 4.1.
05/07/79 00:00   SCMDLIB (BJP)      HCL   RELEASE 4.0.
```

REFERENCES

[1] Ludlam, "Using the DISPLAY-UC Tool". UCLA document UCNSW-107, March 1, 1980.

[2] Ludlam, "Using the UCLA Interim Library Management Tool Kit". UCLA document UCNSW-101, July 18, 1979.